Adding Disjunction to Datalog

- Want to allow disjunction in body of clause
  - $a \leftarrow b \lor c$
  - Means that if either $b$ or $c$ is true, then $a$ must be true
- Not increasing the power of Datalog, just making it easier for knowledge engineer

First, how do we represent definite clauses in Tcl?
- As a list whose first element is head and rest are conjunctions of body

Meta-Interpreters
- How to find proofs in Datalog
  - Depth-first search
  - Breadth-first search
  - A* search
  - Iterative Deepening
- Textbook writes these different interpreters in Prolog
  - Messy, as both base language and metalanguage based on Datalog
  - Base language: what your knowledge engineer programs in
  - Meta-language: what your programmer programs in to implement the reasoning procedure
- We use Tcl for implementing reasoning procedure
  - First, how do we represent definite clauses in Datalog?

Overview
Example

1. When are the new neighbors?
   - Yes → a
   - \{f \lor g\} → a
   - \text{or}
   - \{a \land b\} → a
   - \text{and}
   - \{q \land p\} → a
   - \text{and}
   - \{q \land p\} → a
   - \{p\} → a

2. Interpreter for Disjunction
   - Before (without disjunction)
     - State of a proof was a conjunction
     - Used resolution to replace conjunct \(a\) with a list of conjuncts \(b, c, d\), if we had the rule \(a ← b, c, d\)
   - Now, replacement might be a conjunction or disjunction
     - Conjunction: replace \(a\) with entire conjunction
     - Disjunction: replace \(a\) with one of the disjuncts
     - Note that which disjunct we pick does matter
   - This is not like the don't care non-determinism of picking which conjunct to work on first

3. To find all of the neighbors of \(a\)
   - Need to find all rules whose head matches \(a\)
   - For each rule that is a disjunction, pick all of the disjuncts

Defining Clauses in Tcl
   - Need a more complex representation to handle disjunctions
   - For simplicity, let's not allow embedded bodies. Just:
   { \{ q \land p\} \{ q \land p\} \{ q \land p\} }
Overview

- Disjunction

Depth Bounded

- Delaying Goals

Depth Bounded Reasoning Procedure

- Similar to Iterative Dependent

- Will it always halt?

- Could be done for either depth-first or breadth-first

- But you don't keep going to deeper and deeper depths

- Is it sound and complete?
Rather than always choosing the first atom in the conjunction:

- Have rules for when you can skip over atoms.
- At each step of the proof, keep rechecking whether atoms at the front of the answer clause (which were previously delayed) can be proved.
- Do not move delayed atoms to the end of the answer clause, as you should respect the defined ordering as much as possible.

While we are at it...

- If there are any ground atoms, you might want to prove those first.
- As no variable bindings from earlier atoms will not affect their truth.
- If they can't be proved, may as well find out sooner than later.

Example:

Delaying might allow a definite clause to be used in a way it wasn't anticipated for.

Example: Brother, where we expect both variables to be inputs:

brother(X,Y) :-

- not(X = Y),
- sonof(X,Z),
- sonof(Y,Z),

- When used where either is not input, you get wrong behavior.
- Have reasoning procedure delay evaluation of 'not=', until both parameters are instantiated.
- When used where either is not input, you get wrong behavior.

Delaying Goals

- Some goals, rather than being proved, can be delayed.
- To delay subgoals with variables, in the hope that subsequent calls will ground the variables.
- To delay assumptions, so that you can collect assumptions that are needed to prove a goal.
- We will see more of this later in the course.
Recap of Class

Adding Disjunction
- Did not change expressiveness of Datalog

Depth Bounded Reasoning
- Reasoner produces finite answer in certain amount of time

Reordering of Atoms
- Works in Datalog
- Does not make sense in Prolog
- Does not change expressiveness of Datalog

Caveat
- Reordering of atoms works in Datalog