Beyond Definite Knowledge

• Datalog: Knowledge represented with
  - conjunction of atoms implying something
  - can have variables as well

• Prolog has more than this
  - Syntactic sugar
    + Lists: [a,b|X] versus p(a,p(b,X))
    + Explicit unification: X = Y
  - Not part of the logic:
    + not
    + findall
    + is
    + !
Overview

⇒ Equality

• Reasoning about Equality

• Paramodulation

• Unique Names Assumption
Why Reason About Equality

- Already seen explicit unification
  - Just checks if terms are identical
    + Same constant name
    + Same term expression
- But, we might want more than one term for an object
  - If you have term $motherof(jim)$, you might want to say that that is same term as $mary$
  - Or that Clark Kent is Superman
  - You can do this in your intended interpretation
  - $\phi$ could map two different terms to the same object in domain
Syntax and Equality

• But, don’t have anything in the syntax
  - that will force all interpretations to make two terms the same
    or ensure two terms are different
  - in which we can ask questions about whether two objects are the same

• So, cannot force all models of KB to agree that
  - motherof(jim) is the same as mary
  - jim is not the same mary
Overview

• Equality

⇒ Reasoning about Equality

• Paramodulation

• Unique Names Assumption
• Syntax: $t_1 = t_2$

• Semantics: $I(t_1) = I(t_2)$

• This is much more powerful than Prolog’s ‘=’, which is explicit unification, which is matching symbols from the syntax

• Note: we are not addressing inequality
  - Can be dealt with by adding support for $\neg$

• If we are modeling equality, we should be able to prove this
  
  KB: superman = clarkkent
  male(superman)
  ?male(clarkkent)
Adding to Proof Procedure

• Add axioms
  
  \[ \begin{align*}
  X = X \\
  X = Y & \iff Y = X \\
  X = Z & \iff X = Y \land Y = Z
  \end{align*} \]

• Need axioms for each function symbol
  
  - For each n-ary function symbol \( f \)
    \[ f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n) \iff X_1 = Y_1 \land \ldots \land X_n = Y_n \]
  
  - For each n-ary predicate symbol \( p \)
    \[ p(X_1, \ldots, X_n) \iff p(Y_1, \ldots, Y_n) \land X_1 = Y_1 \land \ldots \land X_n = Y_n \]
Example

• KB
  
motherof(jim) = mary
motherof(john) = mary
member(X,p(X,_))
member(X,p(_,Tail)) ← member(X,Tail)

?member(motherof(john),p(motherof(jim),nil))

• Add axioms
  
X=X
X=Y ← Y=X
X=Z ← X=Y ∧ Y=Z
...

© P. Heeman, 2020
Example

• KB

\[ \text{motherof(jim)} = \text{mary} \]
\[ \text{motherof(john)} = \text{mary} \]
\[ \text{member}(X, p(X, _)) \]
\[ \text{member}(X, p(_, \text{Tail})) \leftarrow \text{member}(X, \text{Tail}) \]
\[ ?\text{member} \left( \text{motherof(john)}, p(\text{motherof(jim)}, \text{nil}) \right) \]

• Add axioms

\[ X=X \]
\[ X=Y \leftarrow Y=X \]
\[ X=Z \leftarrow X=Y \land Y=Z \]
\[ \text{motherof}(X_1) = \text{motherof}(Y_1) \leftarrow X_1=Y_1 \]
\[ p(X_1, X_2) = p(Y_1, Y_2) \leftarrow X_1=Y_1 \land X_2=Y_2 \]
\[ \text{member}(X_1, X_2) \leftarrow \text{member}(Y_1, Y_2) \land X_1=Y_1 \land X_2=Y_2 \]
Proof

\[\text{yes} \leftarrow \text{member}(\text{motherof}(\text{john}), p(\text{motherof}(\text{jim}), \text{nil}))\]
Use \[\text{member}(X_1, X_2) \leftarrow \text{member}(Y_1, Y_2) \land X_1 = Y_1 \land X_2 = Y_2\]
Substitution \[X_1/\text{motherof}(\text{john}) \quad X_2/p(\text{motherof}(\text{jim}), \text{nil})\]

\[\text{yes} \leftarrow \text{member}(Y_1, Y_2) \land \text{motherof}(\text{john}) = Y_1 \land p(\text{motherof}(\text{jim}), \text{nil}) = Y_2\]
Use \[\text{motherof}(\text{john}) = \text{mary}\]
Substitution \[Y_1/\text{mary}\]

\[\text{yes} \leftarrow \text{member}(\text{mary}, Y_2) \land p(\text{motherof}(\text{jim}), \text{nil}) = Y_2\]
Use \[p(X_3, X_4) = p(Y_3, Y_4) \leftarrow X_3 = Y_3 \land X_4 = Y_4\]
Substitution \[X_3/\text{motherof}(\text{jim}) \quad X_4/\text{nil} \quad Y_2/p(Y_3, Y_4)\]

\[\text{yes} \leftarrow \text{member}(\text{mary}, p(Y_3, Y_4) \land \text{motherof}(\text{jim}) = Y_3 \land \text{nil} = Y_4\]
Use \[\text{motherof}(\text{jim}) = \text{mary}\]
Substitution \[Y_3/\text{mary}\]

\[\text{yes} \leftarrow \text{member}(\text{mary}, p(\text{mary}, Y_4) \land \text{nil} = Y_4\]
Use \[X_5 = X_5\]
Substitution \[X_5/\text{nil} \quad Y_4/\text{nil}\]

\[\text{yes} \leftarrow \text{member}(\text{mary}, p(\text{mary}, \text{nil})\]
Use \[\text{member}(X_6, p(X_6, X_7))\]
Substitution \[X_6/\text{mary} \quad X_7/\text{nil}\]

\[\text{yes} \leftarrow \]

© P. Heeman, 2020  9 of 18  CS560 Class 09: Reasoning about Equality
Summary

• Axioms for equality
  - Very inefficient
  - Top-down depth-first interpreter will get stuck
    + For instance, with the symmetrical axiom
Overview

• Equality
• Reasoning about Equality
⇒ Paramodulation
• Unique Names Assumption
Another Approach for Equality

• Have a canonical representation for each domain object
  - Add rewrite rule (paramodulation) to change variant into canonical form

• Example

motherof(jim) = mary  (treated as motherof(jim) ⇒ mary)
motherof(john) = mary  (treated as motherof(john) ⇒ mary)
member(X,p(X,\_))
member(X,p(\_,Tail)) ← member(X,Tail)

?member(motherof(john),p(motherof(jim),nil))

• Proof

yes ← member(motherof(john),p(motherof(jim),nil))
  Paramodulation with motherof(john) ⇒ mary

yes ← member(mary,p(motherof(jim),nil))
  Paramodulation with motherof(jim) ⇒ mary

yes ← member(mary,p(mary,nil))
  Use fact member(X,p(X,Y)) with \{X/mary, Y/nil\}

yes ←
Paramodulation Summary

• No extra equality axioms added to KB
• Equality reasoning only done one way: to rewrite a term with the canonical representation
• Uses a special rewrite mechanism added to theorem prover

• Uses same semantics for equality
  - Is this implementation sound?
  - Is it complete?
Overview

- Equality
- Reasoning about Equality
- Paramodulation

⇒ Unique Names Assumption
Unique Names Assumptions

• Datalog has no mechanism to force two terms to be the same or to force them to be different

• Can add equality
  - Allows us to enforce two terms to be the same
  - But, still can’t force names to be different (since don’t have negation yet)

• But, for certain domains, might want all terms to be different
  - For every pair of ground terms \( t_1 \) and \( t_2 \), assume \( I(t_1) \neq I(t_2) \)
  - Note that this restricts the models of a KB

• Add syntax for stating two things are not the same \( \neq \)

• Semantics of \( \neq \) is simply \( I(t_1 \neq t_2) \) if \( t_1 \neq t_2 \) (from above)
Defining UNA

• Defining inequality with axioms gives way too many axioms
  - A lot more then when we defined equality
  - \( c \neq c' \) for any distinct constants \( c \) and \( c' \)
  - \( f(X_1, \ldots, X_n) \neq g(Y_1, \ldots, Y_m) \) for any distinct function symbols \( f \) and \( g \)
  - \( f(X_1, \ldots, X_n) \neq f(Y_1, \ldots, Y_n) \leftarrow X_i \neq Y_i \) for any function symbol \( f \)
    \( (n \) instances of this rule for each \( f \))
  - \( f(X_i, \ldots, X_n) \neq c \) for any function symbol \( f \) and constant \( c \)
  - \( t \neq X \) for any term \( t \) in which \( X \) appears (where \( t \) is not the term \( X \))
Defining UNA

• Defining inequality with axioms gives way too many axioms
  - A lot more then when we defined equality
  - \( c \neq c' \) for any distinct constants \( c \) and \( c' \)
  - \( f(X_1, \ldots, X_n) \neq g(Y_1, \ldots, Y_m) \) for any distinct function symbols \( f \) and \( g \)
  - \( f(X_1, \ldots, X_n) \neq f(Y_1, \ldots, Y_n) \leftarrow X_i \neq Y_i \) for any function symbol \( f \)
    (\( n \) instances of this rule for each \( f \))
  - \( f(X_i, \ldots, X_n) \neq c \) for any function symbol \( f \) and constant \( c \)
  - \( t \neq X \) for any term \( t \) in which \( X \) appears (where \( t \) is not the term \( X \))

• Our reasoning procedure will explode!
Another Approach

• Build UNA into Top Down Proof Procedure

• \( t_1 \neq t_2 \) succeeds if \( t_1 \) and \( t_2 \) do not unify

• \( t_1 \neq t_2 \) fails if \( t_1 \) and \( t_2 \) are identical

• Otherwise, if \( t_1 \) and \( t_2 \) can unify
  
  - There are variables involved: some instances succeed and some fail
  
  - Could enumerate every instance that causes goal to succeed (all ways of making \( t_1 \) and \( t_2 \) different), but way too many...

  - Instead, delay the goal

  - If can’t be delayed anymore, goal should succeed, but be careful of how free variables in query are interpreted

    + Means that it succeeds for some values of the free variable, but not necessarily for all
Contrast to Prolog’s ‘=’

• For \( \text{not}(t_1 = t_2) \)
  - Prolog succeeds if they don’t unify
  - Otherwise it fails
  - It doesn’t delay the goal where it is unsure

• So, if you are careful where you place \( \text{not}(t_1 = t_2) \) in clauses (so that all variables are bound), this gives you the UNA assumption for Prolog

• For a lot of domains, natural to assume UNA
  - Homework 4: building block tower