
Beyond Definite Knowledge

- **Datalog: Knowledge represented with**
 - conjunction of atoms implying something
 - can have variables as well
- **Prolog has more than this**
 - Has 'not'
 - Has 'findall'
 - Lists don't add any expressive power
 - Explicit unification does not add any power

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
 - ϕ 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

Add to Syntax

- 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 that this is not addressing inequality
 - Can be dealt with by adding support for \neq

Adding to Proof Procedure

- Add axioms
 - $X = X$
 - $X = Y \leftarrow Y = X$
 - $X = Z \leftarrow X = Y \wedge Y = Z$
- Need axioms for each function symbol
 - For each n-ary function symbol f

$$f(X_1, \dots, X_n) = f(Y_1, \dots, Y_n) \leftarrow X_1 = Y_1 \wedge \dots \wedge X_n = Y_n$$
 - For each n-ary predicate symbol p

$$p(X_1, \dots, X_n) \leftarrow p(Y_1, \dots, Y_n) \wedge X_1 = Y_1 \wedge \dots \wedge X_n = Y_n$$

Example

- KB
 - $motherof(jim) = mary$
 - $motherof(john) = mary$
 - $member(X, cons(X, _))$
 - $member(X, cons(_, Tail)) \leftarrow member(X, Tail)$
 - $?member(motherof(john), cons(motherof(jim), nil))$
- Add axioms
 - $X = X$
 - $X = Y \leftarrow Y = X$
 - $X = Z \leftarrow X = Y \wedge Y = Z$
 - $motherof(X_1) = motherof(Y_1) \leftarrow X_1 = Y_1$
 - $cons(X_1, X_2) = cons(Y_1, Y_2) \leftarrow X_1 = Y_1 \wedge X_2 = Y_2$
 - $member(X_1, X_2) \leftarrow member(Y_1, Y_2) \wedge X_1 = Y_1 \wedge X_2 = Y_2$

Proof

yes ← *member*(*motherof*(*john*),*cons*(*motherof*(*jim*),*nil*))
 Use $\text{member}(X_1, X_2) \leftarrow \text{member}(Y_1, Y_2) \wedge X_1 = Y_1 \wedge X_2 = Y_2$
 Substitution $X_1/\text{motherof}(\text{john}) \ X_2/\text{cons}(\text{motherof}(\text{jim}), \text{nil})$
yes ← *member*(Y_1, Y_2) ∧ *motherof*(*john*) = Y_1 ∧ *cons*(*motherof*(*jim*), *nil*) = Y_2
 Use $\text{motherof}(\text{john}) = \text{mary}$
 Substitution Y_1/mary
yes ← *member*(*mary*, Y_2) ∧ *cons*(*motherof*(*jim*), *nil*) = Y_2
 Use $\text{cons}(X_3, X_4) = \text{cons}(Y_3, Y_4) \leftarrow X_3 = Y_3 \wedge X_4 = Y_4$
 Substitution $X_3/\text{motherof}(\text{jim}) \ X_4/\text{nil} \ Y_3/\text{cons}(Y_3, Y_4)$
yes ← *member*(*mary*, *cons*(Y_3, Y_4) ∧ *motherof*(*jim*) = Y_3 ∧ *nil* = Y_4
 Use $\text{motherof}(\text{jim}) = \text{mary}$
 Substitution Y_3/mary
yes ← *member*(*mary*, *cons*(*mary*, Y_4) ∧ *nil* = Y_4
 Use $X_5 = X_5$
 Substitution $X_5/\text{nil} \ Y_4/\text{nil}$
yes ← *member*(*mary*, *cons*(*mary*, *nil*))
 Use $\text{member}(X_6, \text{cons}(X_6, X_7))$
 Substitution $X_6/\text{mary} \ X_7/\text{nil}$
yes ←

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
- Rewrite all variants with the canonical representation
 - Approach called paramodulation

- **Example**

motherof(jim) = mary

motherof(john) = mary

member(X,p(X,-))

member(X,p(-,Tail)) ← member(X,Tail)

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

- **Proof**

Let *mary* be the canonical representation for *motherof(jim)* and *motherof(john)*

yes ← *member(motherof(john),p(motherof(jim),nil))*

yes ← *member(mary,p(mary,nil))*

yes ←

Summary

- No extra equality relations 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 than when we defined equality
 - $c \neq c'$ for any distinct constants c and c'
 - $f(X_1, \dots, X_n) \neq g(Y_1, \dots, Y_m)$ for any distinct function symbols f and g
 - $f(X_1, \dots, X_n) \neq f(Y_1, \dots, Y_n) \leftarrow X_i \neq Y_i$ for any function symbol f (n instances of this rule for each f)
 - $f(X_i, \dots, X_n) \neq c$ 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 they don't 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 *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 *not*($t_1 = t_2$) in clauses (so that all variables are bound), this gives you the UNA assumption for Prolog

Usefulness of UNA

- **Homework 4**
 - Want to build a tower of different blocks
 - You had to define a special 'different' predicate
 - Would have been nice to assume the UNA assumption
- **Finding out if two people are brothers**
 - Same mother, but different people