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 Jeff is same
  - You could have terms like identical
  - Just check if terms are identical

Already seen explicit unification

Overview

Beyond Definite Knowledge

• Datalog: Knowledge represented with
  - Conjunction of atoms implying something
  - Can have variables as well

• Prolog has more than this
  - Can have variables well
  - Can be used to implement something
  - Datalog: Knowledge represented with
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

Overview

• Equality
  ⇒ Reasoning about Equality
• Paramodulation
  Unique Names Assumption

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.
• But don't have anything in the syntax.
Proof

Example

KB

Adding to Proof Procedure
• Add axioms

- For each n-ary function symbol

- For each n-ary predicate symbol

Real axiom for each function symbol

Add axioms •

CSE560 Class 09: 9 c
P. Heeman, 2010
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
- motherof(john) = mary
- member(X,p(X,Y))
- member(X,p(Y,Tail))

Proof:

- ?member(motherof(john),p(motherof(jim),nil))
  + Paramodulation with motherof(john) ⇒ mary
  + Paramodulation with motherof(jim) ⇒ mary
  + Use fact member(X,p(X,Y)) with {X/mary, Y/nil}

Overview:

- Equality
- Reasoning about Equality
  + Paramodulation
- Unique Names Assumption

Summary:

- For instance, with the symmetrical axiom:
  - Top-down depth-first interpreter will get stuck
- Very inefficient
- Axioms for equality
Unique Names Assumptions

- Equality is simply \( f(t_1) \neq f(t_2) \) (from above)
- No extra axioms to force two things are not the same
- In some cases, this restricts the models of a KB
- For every pair of ground terms \((t_1, t_2)\) and \((t'_1, t'_2)\), assume \( f(t_1) \neq f(t_2) \)
- But for certain domains, might want all terms to be different
- flair will canonize terms to be different (since don't have negation yet)
- Allows us to enforce two terms to be the same
- Can add equality to force them to be different
- But does this no additional to force two terms to be the same or different

\[
\begin{align*}
\text{Unique Names Assumption} \\
\text{Paramodulation} \\
\text{Equality reasoning} \\
\text{Equality}
\end{align*}
\]

Overview

- Is it complete?
- Is this implementation sound?
- Uses same semantics for equality
- Uses a special rewrite mechanism added to theorem prover
- The canonical representation is relatively easy to rewrite a lemma with
- Equality reasoning only done one way: to rewrite a lemma with
- No extra equality axioms added to KB
Contrast to Prolog's equality

- For not(t₁ = t₂)
  - 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(t₁ = t₂) in clauses
- Prolog succeeds if they don't unify
- For a lot of domains, natural to assume UNA

Another Approach

- Build UNA into Top Down Proof Procedure
- t₁ ≠ t₂ succeeds if t₁ and t₂ do not unify
- t₁ ≠ t₂ fails if t₁ and t₂ are identical
- Otherwise, if they can unify
- There are variables involved: some instances succeed and some fail

Our reasoning procedure will explode

Defining UNA

- Defining inequality with axioms gives way too many axioms