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 just one term for an object
  - If you have term motherof(jim), you might want to say that term is the same as
  - 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

---

Overview

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

---

Beyond Definite Knowledge

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

- Prolog has more
  - Has 'not'
  - Has 'findall'
  - Lists don't add any expressive power
  - Explicit unification does not add any power
Add to 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 ¬.
Proof

Example

Adding to Proof Procedure

KB • Adding to Proof Procedure

KB • Adding to Proof Procedure

KB • Adding to Proof Procedure
Another Approach for Equality

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

    motherof(jim) = mary (⇒ motherof(jim) ⇒ mary)
    motherof(john) = mary (⇒ motherof(john) ⇒ mary)

  member(X,p(X, Tail))
  ← member(X, p(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
    }

Overview

• Unique Names Assumption
  = Paramodulation
  ⇒ Reasoning about Equality
    Equality

Summary

• For instance, with the symmetrical axiom
  + Very efficient (grounded form): changes variants into canonical form
  + Have a canonical representation for each domain object
Unique Names Assumptions

• Datalog has no mechanism to force two terms to be the same.
  - Can add equality, allowing 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 \( f(t_1) \neq f(t_2) \) from above.

Overview

• Equality
• Reasoning about Equality
• Paramodulation

Summary

• No extra equality axioms added to KB.
• Uses equality reasoning only where one way to rewrite a term with equality expression only does one way to rewrite a term with equality expression only.
• Uses paramodulation only where one way to rewrite a term with equality expression only does one way to rewrite a term with equality expression only.
• No extra equality axioms added to KB.
Contrast to Prolog's `='

- 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
  - (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

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 `t1` and `t2` can unify
  - There are variables involved: some instances succeed and some fail
  - Theorems: `t1 ≠ t2`
  - Build UNA into Top Down Proof Procedure

Defining UNA

- Necessary for all
  - `c ≠ c'` for any distinct constants `c` and `c'`
  - `f(X1, ..., Xn) ≠ g(Y1, ..., Ym)` for any distinct function symbols `f` and `g`
  - `f(X1, ..., Xn) ≠ f(Y1, ..., Yn)` ← `Xi ≠ Yj` for any function symbol `f`
  - `f(X1, ..., Xn) ≠ c` for any function symbol `f` and constant `c`
- Our reasoning procedure will explode!