Why Reason About Equality

- Already seen explicit unification
  - Just checks if terms are identical
- 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
  - You can do this in your intended interpretation
  - \( \phi \) could map two different terms to the same object in domain
- Can do this in your intended interpretation
- Even if many
- If you have term motherof(jim), you might want to say that that is same
- But we might want more than one term for an object
- Already seen explicit unification

Overview

- 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 than this
  - Syntactic sugar
    - Lists: \([a, b | X] \) versus \( p(a, p(b, X)) \)
    - Explicit unification: \( X = Y \)
      - Not part of the logic
- Not part of the logic
  -\\( A = B [\text{expression: } \lambda x . \text{expression: } p(x)] \)
  - Lambda abstraction
  - Default negation
- Default negation
- Default negation

- Default negation

- Default negation
  - Default negation
  - Default negation
  - Default negation

- Default negation
  - Default negation
  - Default negation
  - Default negation

- Default negation
  - Default negation
  - Default negation
  - Default negation

- Default negation
  - Default negation
  - Default negation
Add to Syntax

- `t_1 = t_2`
  - Syntax:
  - `t_1 = t_2`
  - Semantics:
    - `I(t_1) = I(t_2)`
  - This is much more powerful than Prolog's `=`.
  - 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

- Can force all models of KB to agree that
  - two terms are the same
  - two terms are different
- But, don't force anything in the syntax.
Proof

Example

Adding to Proof Procedure

KB

Real axioms for each function symbol

Real axioms for each function symbol

Add axioms

Add axioms
Another Approach for Equality

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

  Example
  \[
  \text{motherof}(jim) = \text{mary} \\quad (\text{treated as } \text{motherof}(jim) \Rightarrow \text{mary})
  \]
  \[
  \text{motherof}(john) = \text{mary} \\quad (\text{treated as } \text{motherof}(john) \Rightarrow \text{mary})
  \]

  \[
  \text{member}(X, p(X, \text{Tail}))
  \]
  \[
  \text{member}(X, p(\text{Tail}))
  \]

  \[
  ?\text{member}(\text{motherof}(john), p(\text{motherof}(jim), \text{nil}))
  \]

• Proof
  yes
  \[
  \text{paramodulation with } \text{motherof}(john) \Rightarrow \text{mary}
  \]
  yes
  \[
  \text{paramodulation with } \text{motherof}(jim) \Rightarrow \text{mary}
  \]
  yes
  \[
  \text{use fact } \text{member}(X, p(X, \text{Y}))
  \]
  with
  \[
  \{X/\text{mary}, Y/\text{nil}\}
  \]

Overview

• Equality
• Reasoning about Equality
  \[
  \text{Paramodulation}
  \]
  \[
  \text{Unique Names Assumption}
  \]

Summary

- Axioms for equality
  - For instance, with the symmetric axiom
  - Top-down depth-first interpreter will not such
  - Very inefficient
  - Use fact
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, 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).
- For certain domains, might want all terms to be different.
- But still can't force names to be different (since don't have negation yet).
- Can add equality to force them to be different.
- No extra equality axioms added to KB.
- Equality reasoning only done one way: to rewrite a term with the canonical representation.
- Uses same semantics for equality.
- Uses a special rewrite mechanism added to theorem prover.
- Is it complete?
- Is this implementation sound?
- Less same semantics for equality.

Summary
Contrast to Prolog's `=`

- For `not(t1 = t2)`
  - Prolog succeeds if they don't unify
  - Otherwise, it fails
  - It doesn't delay the goal when 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

Another Approach

- Build UNA into Top Down Proof Procedure
- `t1 \neq t2` succeeds if `t1` and `t2` do not unify
- `t1 \neq t2` fails if `t1` and `t2` are identical
- Otherwise, if `t1` and `t2` can unify
  - if `t1 \neq t2` and `t2` is identical
    - ` succeeds if and only if `t1 \neq t2`
    - ` fails if and only if `t1 \neq t2`
- Build UNA into Top Down Proof Procedure

Our reasoning procedure will explode!

- For all terms `t` where `t` appears in a term `s` (where `t` is not the head `X`)
- for any function symbol `f` and constant `c`
  - if `t` is the head of the term `f`
    - if `f \neq X` and `t` appears in `f`.
      - `X \neq \ldots \neq X`
    - for any function symbol `g` and constant `c`
      - if `g \neq X` and `t` appears in `g`
        - `X \neq \ldots \neq X`

Defining inequality with axioms gives way too many axioms