



---

# Automated Reasoning Support for First-Order Ontologies

presented at the  
4<sup>th</sup> Workshop on Principles and Practice  
of Semantic Web Reasoning (PPSWR 2006)

Peter Baumgartner (National ICT Australia, Canberra/Australia)

Fabian M. Suchanek (Max-Planck-Institute for CS, Saarbruecken/Germany)

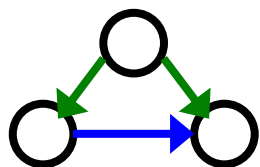
# Automated Reasoning Support for FOLs



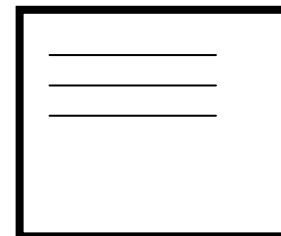
MAX-PLANCK-GESELLSCHAFT

- ♪ Motivation
- ♪ Our Transformation from FOL to DLPs
  - ♪ Existentially Quantified Formulae
  - ♪ Equality
- ♪ Conclusion

# Model Computation



Ontology  
(set of FOL formulae)



Model  
(set of derivable facts)

$\forall x \text{ singer}(x) \Rightarrow \text{sings}(x)$

$\text{singer}(\text{elvis})$

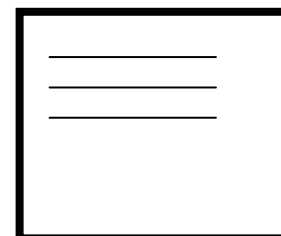
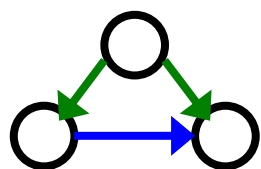
$\text{sings}(\text{elvis})$ .

$\text{singer}(\text{elvis})$ .

# Use of Model Computation



MAX-PLANCK-GESELLSCHAFT



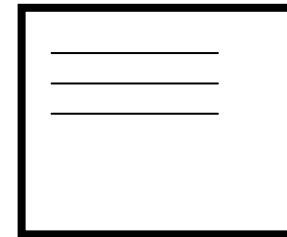
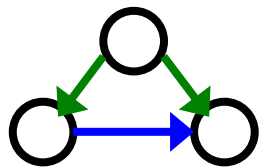
Model computation can be used for:

- ♪ Finding contradictions in the ontology  
(there is a model only iff the ontology is consistent)
- ♪ Debugging the ontology
- ♪ Proving/disproving conjectures

# Undecidability of Model Computation



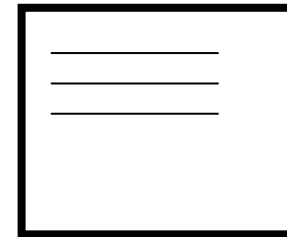
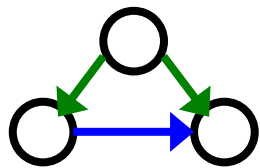
MAX-PLANCK-GESELLSCHAFT



Model Computation is only semi-decidable for FOL:

- ⌋ we can (in principle) always detect unsatisfiability
- ⌋ we cannot always detect satisfiability (for principal reasons)

# Existing Approaches



There exist model generation systems (e.g. MACE4 and Paradox).

Problems:

- ⌋ They try to map all constants to the same domain element
- ⌋ They have difficulties if there are many distinct constants

$$p(c_1, \dots, c_n).$$

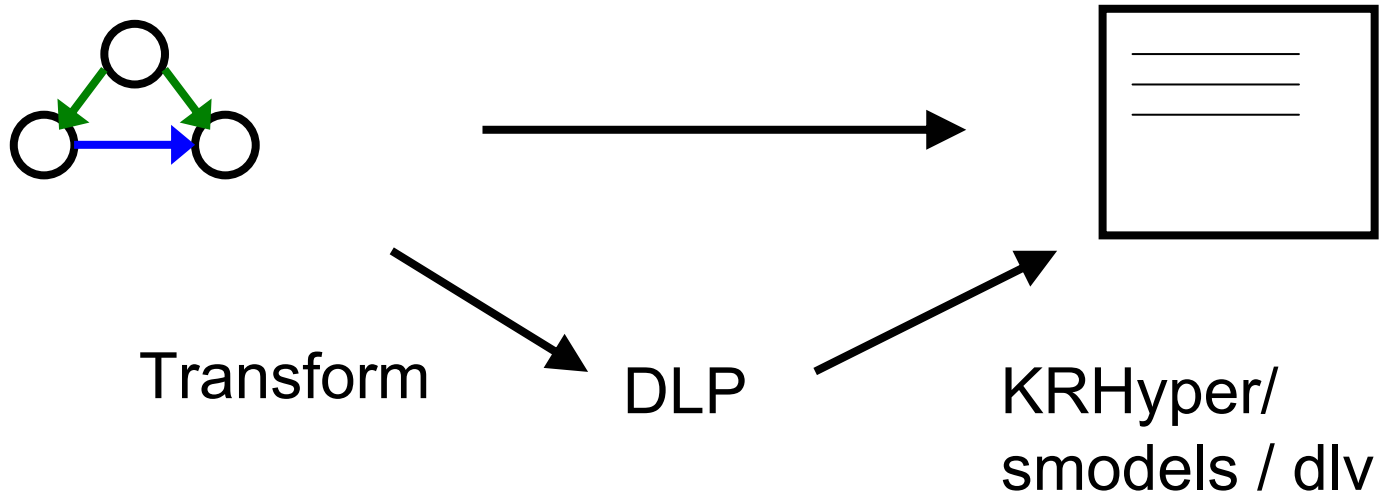
$$\neg p(x_1, \dots, x_{i-1}, \mathbf{x}, x_{i+1}, \dots, x_{j-1}, \mathbf{x}, x_{j+1}, \dots, x_n). \quad \text{for all } 1 < i < j < n$$

**Fails for  
n > 8**

# Our Approach



MAX-PLANCK-GESELLSCHAFT



# Disjunctive Logic Programs (DLPs)



MAX-PLANCK-GESELLSCHAFT

Rule:

$$r(a) \vee p(x,y) \text{ :- } q(x,y), r(z), \text{ not}(s(a,x)).$$

A Disjunctive Logic Program (DLP) is a set of rules.

# Our Transformation from FOL to DLPs



MAX-PLANCK-GESellschaft

$\forall x \text{ inCharge}(x) \Rightarrow \text{onLeave}(x) \vee \exists y \text{ refersTo}(x,y)$



**Prenex Negation Normal Form**

$\forall x[\dots] \exists y[\dots] [Qz\dots] \neg \text{inCharge}(x) \vee \text{onLeave}(x) \vee \text{refersTo}(x,y)$

# Our Transformation from FOL to DLPs



MAX-PLANCK-GESellschaft

$$\forall x \text{ inCharge}(x) \Rightarrow \text{onLeave}(x) \vee \exists y \text{ refersTo}(x,y)$$

$$\forall x[\dots] \exists y[\dots] [Qz\dots] \text{inCharge}(x) \vee \text{onLeave}(x) \vee \text{refersTo}(x,y)$$

# Our Transformation from FOL to DLPs



MAX-PLANCK-GESellschaft

$$\forall x \text{ inCharge}(x) \Rightarrow \text{onLeave}(x) \vee \exists y \text{ refersTo}(x, y)$$

$$\forall x[\dots] \exists y[\dots] [Qz\dots] \neg \text{inCharge}(x) \vee \text{onLeave}(x) \vee \text{refersTo}(x, y)$$

Disjuncts without  
existential or  
following variables

Disjuncts with  
existential or  
following variables

# Our Transformation from FOL to DLPs



MAX-PLANCK-GESELLSCHAFT

$$\forall x \text{ inCharge}(x) \Rightarrow \text{onLeave}(x) \vee \exists y \text{ refersTo}(x,y)$$



$$\forall x[\dots] \exists y[\dots] [Qz\dots] \neg \text{inCharge}(x) \vee \text{onLeave}(x) \vee \text{refersTo}(x,y)$$

Disjuncts without  
existential or  
following variables

Disjuncts with  
existential or  
following variables

# Our Transformation from FOL to DLPs



MAX-PLANCK-GESellschaft

$$\forall x \text{ inCharge}(x) \Rightarrow \text{onLeave}(x) \vee \exists y \text{ refersTo}(x,y)$$



negative literals

$$\forall x[\dots] \exists y[\dots] [Qz\dots] \neg \text{inCharge}(x) \vee \text{onLeave}(x) \vee \text{refersTo}(x,y)$$

Disjuncts without  
existential or  
following variables

Disjuncts with  
existential or  
following variables

# Our Transformation from FOL to DLPs



MAX-PLANCK-GESELLSCHAFT

$\forall x \exists y \neg inCharge(x) \vee \underline{onLeave(x)} \vee \underline{refersTo(x,y)}$



Option 1: Usual Skolemization

$\underline{onLeave(x)} \vee \underline{refersTo(x,sk(x))} :- inCharge(x).$

# Our Transformation from FOL to DLPs

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 1: Usual Skolemization

$$\underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x, \text{sk}(x))} \text{ :- } \text{inCharge}(x).$$

Problem:

$\neg \text{onLeave}(\text{Smith})$

$\text{refersTo}(\text{Smith}, \text{Miller})$

$\rightarrow \text{refersTo}(\text{Smith}, \text{sk}(\text{Smith})).$

# Our Transformation from FOL to DLPs

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 2: Recycling

$$\underline{\text{is\_sat}_\exists(x)} \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,\text{sk}(x))} \text{ :- inCharge}(x).$$

Problem:

$\neg \text{onLeave}(\text{Smith})$

$\text{refersTo}(\text{Smith}, \text{Miller})$

$\rightarrow \text{refersTo}(\text{Smith}, \text{sk}(\text{Smith})).$

(simplified. See paper)

# Our Transformation from FOL to DLPs

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 2: Recycling

$$\underline{\text{is\_sat}_\exists(x)} \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,\text{sk}(x))} \text{ :- inCharge}(x).$$

false :- is\_sat<sub>∃</sub>(x),  
not(refersTo(x,y)).

Problem:

$\neg \text{onLeave}(\text{Smith})$

$\text{refersTo}(\text{Smith}, \text{Miller})$

→  $\text{refersTo}(\text{Smith}, \text{sk}(\text{Smith}))$ .

(simplified. See paper)

# Our Transformation from FOL to DLPs

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 2: Recycling

$$\underline{\text{is\_sat}_\exists(x)} \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,\text{sk}(x))} \text{ :- inCharge}(x).$$

false :- is\_sat<sub>∃</sub>(x),  
not(refersTo(x,y)).

Problem:

$\neg \text{onLeave}(\text{Smith})$

$\text{refersTo}(\text{Smith}, \text{Miller})$

→ ~~refersTo~~(Smith, sk(Smith)).

(simplified. See paper)

# Our Transformation from FOL to DLPs



MAX-PLANCK-GESellschaft

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 2: Recycling

$$\underline{\text{is\_sat}_\exists(x)} \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,\text{sk}(x))} \text{ :- inCharge}(x).$$

false :- is\_sat<sub>∃</sub>(x),

not(refersTo(x,y)).

Problem:

$$\neg \exists x \text{ onLeave}(x)$$

→ refersTo(Smith,sk(Smith)).

# Our Transformation from FOL to DLPs



MAX-PLANCK-GESellschaft

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 2: Recycling

$$\underline{\text{is\_sat}_\exists(x)} \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,\text{sk}(x))} \text{ :- inCharge}(x).$$

false :- is\_sat<sub>∃</sub>(x),

not(refersTo(x,y)).

Problem:

$$\neg \exists x \text{ onLeave}(x)$$

→ refersTo(Smith,sk(Smith)).

refersTo(sk(Smith),sk(sk(Smith))).

refersTo(sk(sk(Smith)),sk(sk(sk(Smith)))).

refersTo(sk(sk(sk(Smith))),sk(sk(sk(sk(Smith))))).

refersTo(sk(sk(sk(sk(Smith))),sk(sk(sk(sk(sk(Smith)))))).



# Our Transformation from FOL to DLPs

refersTo(sk(sk(sk(sk(sk(Smith))),sk(sk(sk(sk(sk(sk(Smith)))))).

refersTo(sk(sk(sk(sk(sk(sk(Smith))))),sk(sk(sk(sk(sk(sk(sk(Smith)))))).

ersTo(sk(sk(sk(sk(sk(sk(sk(Smith))))),sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))).

$\forall x \exists y \neg inCharge(x) \vee \underline{onLeave(x)} \vee \underline{refersTo(x,y)}$

k(sk(sk(sk(sk(sk(sk(sk(Smith))))),sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))).

## Option 2: Recycling

(sk(sk(sk(sk(sk(Smith))))),sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))).

sk(sk(sk(sk(sk(Smith))))),sk(sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))).

$is\_sat_{\exists}(x) \vee onLeave(x) \vee refersTo(x,sk(x)) \text{ :- } inCharge(x).$

sk(sk(sk(sk(Smith))))),sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))).

false :- is\_sat\_{\exists}(x),

sk(sk(sk(Smith))))),sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))).

not(refersTo(x,y)).

Problem:

k(sk(Smith))))),sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))).

$\neg \exists x onLeave(x)$

sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))) → refersTo(Smith,sk(Smith)).

sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))) refersTo(sk(Smith),sk(sk(Smith))).

sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(Smith)))))) refersTo(sk(sk(Smith)),sk(sk(sk(Smith))).

refersTo(sk(sk(sk(Smith))),sk(sk(sk(sk(Smith))))).

# Our Transformation from FOL to DLPs

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 2: Recycling

$$\underline{\text{is\_sat}_\exists(x)} \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,\text{sk}(x))} \text{ :- inCharge}(x).$$

false :- is\_sat<sub>∃</sub>(x),

not(refersTo(x,y)).

Problem:

$$\neg \exists x \text{ onLeave}(x)$$

→ refersTo(Smith,sk(Smith)).

(simplified. See paper)

# Our Transformation from FOL to DLPs

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 3: Loop check

$$\underline{\text{prev}_{\exists}(x)} \vee \underline{\text{is\_sat}_{\exists}(x)} \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,\text{sk}(x))} \text{ :- inCharge}(x).$$

Problem:

$$\neg \exists x \text{ onLeave}(x)$$

→  $\text{refersTo}(\text{Smith}, \text{sk}(\text{Smith}))$ .

(simplified. See paper)

# Our Transformation from FOL to DLPs

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 3: Loop check

$$\underline{\text{prev}_{\exists}(x)} \vee \underline{\text{is\_sat}_{\exists}(x)} \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,\text{sk}(x))} \text{ :- inCharge}(x).$$

false :- is\_sat<sub>∃</sub>(x),  
not(refersTo(x,y)).

refersTo(x,sk(z)) :-  
prev<sub>∃</sub>(x),  
refersTo(z,sk(z)).

Problem:

$$\neg \exists x \text{ onLeave}(x)$$

→ refersTo(Smith,sk(Smith)).

(simplified. See paper)

# Our Transformation from FOL to DLPs

$$\forall x \exists y \neg \text{inCharge}(x) \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,y)}$$


Option 3: Loop check

$$\underline{\text{prev}_{\exists}(x)} \vee \underline{\text{is\_sat}_{\exists}(x)} \vee \underline{\text{onLeave}(x)} \vee \underline{\text{refersTo}(x,\text{sk}(x))} \text{ :- inCharge}(x).$$

false :- is\_sat<sub>∃</sub>(x),  
not(refersTo(x,y)).

refersTo(x,sk(z)) :-  
prev<sub>∃</sub>(x),  
refersTo(z,sk(z)).

Problem:

$$\neg \exists x \text{ onLeave}(x)$$

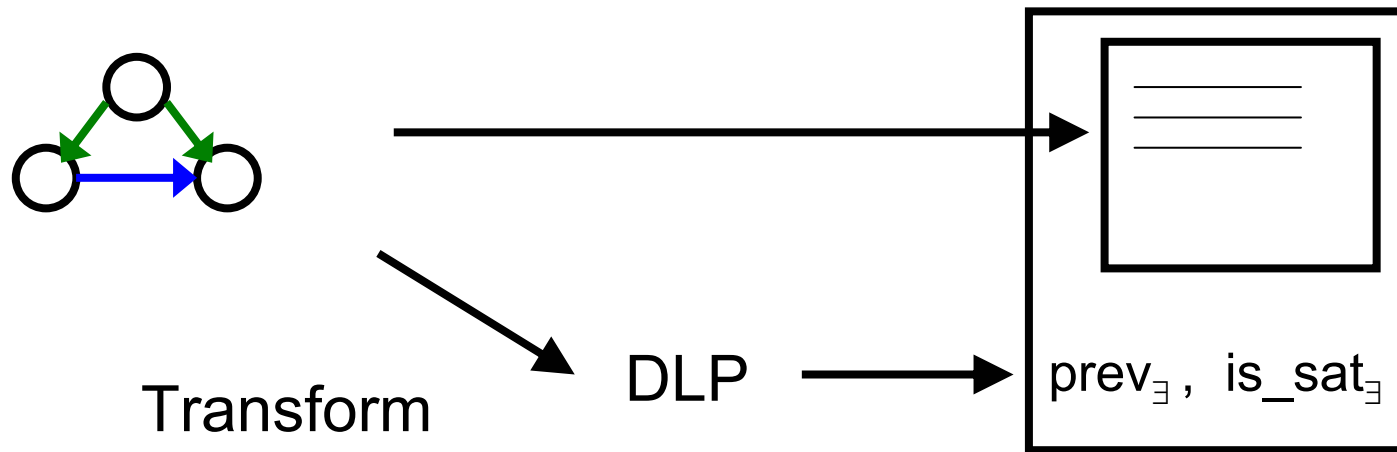
→ refersTo(Smith,sk(Smith)).  
refersTo(sk(Smith),sk(Smith)).

(simplified. See paper)

# Our Transformation from FOL to DLPs



MAX-PLANCK-GESELLSCHAFT



The model is preserved (modulo the fresh predicate symbols).

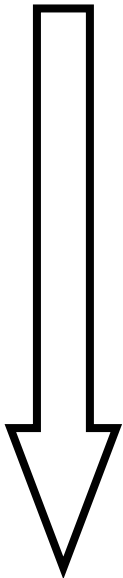
# Treating Equality



MAX-PLANCK-GESELLSCHAFT

`equal(dog(smith), bonzo).`

`inCharge(dog(smith)).`



`inCharge(bonzo).`

# Treating Equality



MAX-PLANCK-GESellschaft

`equal(dog(smith), bonzo).`

`inCharge(dog(smith)).`

|  |   |                        |
|--|---|------------------------|
| <code>inCharge(Y):-inCharge(X),equal(X,Y).</code>  | } | Substitution<br>axioms |
| <code>equal(dog(X),dog(Y)) :- equal(X,Y).</code>   |   |                        |
| <code>equal(X,X).</code>                           | } | Equivalence<br>axioms  |
| <code>equal(X,Y) :- equal(Y,X).</code>             |   |                        |
| <code>equal(X,Z) :- equal(X,Y), equal(Y,Z).</code> |   |                        |

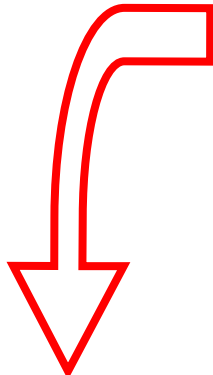
# Treating Equality



MAX-PLANCK-GESellschaft

`equal(dog(smith), bonzo).`

`inCharge(dog(smith)).`



`inCharge(Y):-inCharge(X),equal(X,Y)` } Substitution  
`equal(dog(X),dog(Y)) :- equal(X,Y).` } axioms  
`equal(X,X).` }  
`equal(X,Y) :- equal(Y,X).` } Equivalence  
`equal(X,Z) :- equal(X,Y), equal(Y,Z).` } axioms

`equal(dog(dog(smith)),dog(bonzo)).`



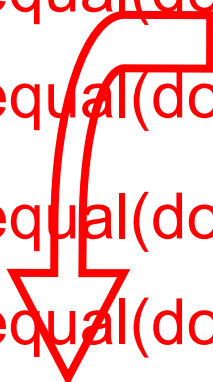
# Treating Equality

equal(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(elvi  
 equal(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(smith))))))  
 equal(dog(smith), bonzo)).

inCharge(dog(dog(smith))).

equal(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(smith)))))))))  
 inCharge(Y):-inCharge(X),equal(X,Y) } Substitution  
 equal(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(smith)))))))))  
 equal(dog(X),dog(Y)) :- equal(X,Y). } axioms  
 equal(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(wif  
 equal(X,X). }  
 equal(dog(dog(dog(dog(dog(dog(smith))))))  
 equal(X,Y) :- equal(Y,X). } Equivalence  
 equal(dog(dog(dog(dog(dog(dog(smith))))  
 equal(X,Z) :- equal(X,Y), equal(Y,Z). } axioms  
 equal(dog(dog(dog(dog(dog(dog(smith))))  
 equal(dog(dog(dog(smith))),dog(dog(bonzo))).

equal(dog(dog(smith)),dog(bonzo)).



# Treating Equality



MAX-PLANCK-GESELLSCHAFT

`equal(dog(smith), bonzo).`

`inCharge(dog(smith)).`

|  |   |                        |
|--|---|------------------------|
| <code>inCharge(Y):-inCharge(X),equal(X,Y).</code>  | } | Substitution<br>axioms |
| <code>equal(dog(X),dog(Y)) :- equal(X,Y).</code>   |   |                        |
| <code>equal(X,X).</code>                           | } | Equivalence<br>axioms  |
| <code>equal(X,Y) :- equal(Y,X).</code>             |   |                        |
| <code>equal(X,Z) :- equal(X,Y), equal(Y,Z).</code> |   |                        |

# Treating Equality



MAX-PLANCK-GESellschaft

`equal(dog(smith), bonzo).`

`inCharge(dog(smith)).`

`equal(X,X).`

`equal(X,Y) :- equal(Y,X).`

`equal(X,Z) :- equal(X,Y), equal(Y,Z).`

} Equivalence  
axioms

# Treating Equality



MAX-PLANCK-GESELLSCHAFT

equal(dog(smith), bonzo).

inCharge(  $X$  ) :- equal( $X$ , dog(smith)).

Flatten function terms (Brand 1975)

equal( $X$ , $X$ ).

equal( $X$ , $Y$ ) :- equal( $Y$ , $X$ ).

equal( $X$ , $Z$ ) :- equal( $X$ , $Y$ ), equal( $Y$ , $Z$ ).

} Equivalence axioms

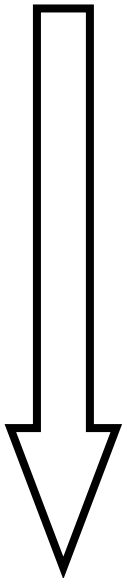
# Treating Equality



MAX-PLANCK-GESELLSCHAFT

`equal(dog(smith), bonzo).`

`inCharge( X ) :- equal(X,dog(smith)).`



`equal(X,X).`

`equal(X,Y) :- equal(Y,X).`

`equal(X,Z) :- equal(X,Y), equal(Y,Z).`

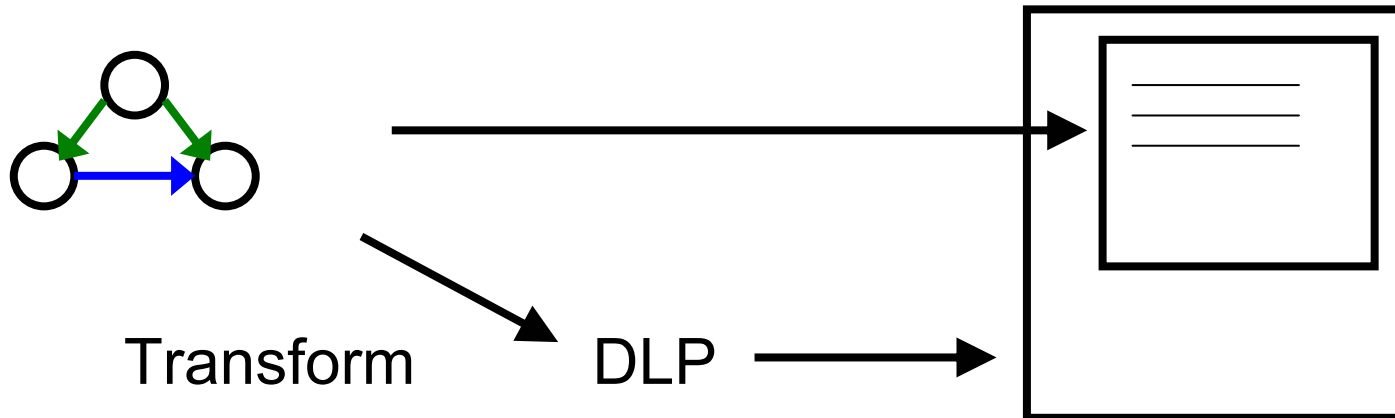
} Equivalence  
axioms

`inCharge(bonzo).`

# Treating Equality



MAX-PLANCK-GESELLSCHAFT

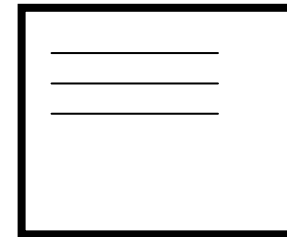
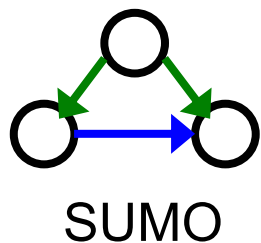


Iff the original DLP has a model that satisfies equality,  
then the transformed DLP has a model (which contains the original one)

# Preliminary Experiments with SUMO



MAX-PLANCK-GESELLSCHAFT



The Suggested Upper Merged Ontology (SUMO):

- ⌋ is the largest public formal ontology available today
- ⌋ uses first order logic with higher order features
- ⌋ contains 1800 facts and rules if higher order features are stripped

$$\forall x \text{ human}(x) \Rightarrow \exists y \text{ mother}(x,y)$$

# Preliminary Experiments with SUMO



MAX-PLANCK-GESellschaft

Applying our approach to SUMO:

- ♪ The transformation to a DLP takes just a few seconds
- ♪ The model computation with KRHyper takes just a few seconds
- ♪ The model computation revealed numerous inconsistencies in SUMO

# Preliminary Experiments with SUMO



MAX-PLANCK-GESELLSCHAFT

Results:

- └ Equality transformation proved scalable and useful

# Preliminary Experiments with SUMO



MAX-PLANCK-GESELLSCHAFT

## Results:

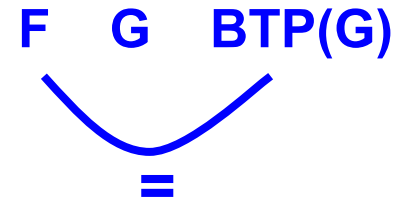
- Equality transformation proved scalable and useful

We added the following conjectures to SUMO:

*orientation(germany,west, biggestTradingPartner(germany)).*

*orientation(france, west, germany).*

*equal(biggestTradingPartner(germany),france).*



# Preliminary Experiments with SUMO

## Results:

- Equality transformation proved scalable and useful

We added the following conjectures to SUMO:

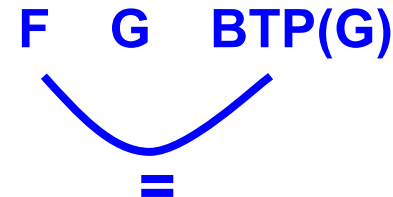
*orientation(germany,west, biggestTradingPartner(germany)).*

*orientation(france, west, germany).*

*equal(biggestTradingPartner(germany),france).*

*orientation(germany, east, france).*

*orientation(germany, west, france).* 



# Preliminary Experiments with SUMO



MAX-PLANCK-GESELLSCHAFT

## Results:

- ⌋ Equality transformation proved scalable and useful
- ⌋ Recycling of terms works as expected

# Preliminary Experiments with SUMO



MAX-PLANCK-GESellschaft

## Results:

- ⌋ Equality transformation proved scalable and useful
- ⌋ Recycling of terms works as expected

We added the following conjecture to SUMO:

*instance(p, judicialProcess)*

# Preliminary Experiments with SUMO



MAX-PLANCK-GESellschaft

Results:

- ✓ Equality transformation proved scalable and useful
- ✓ Recycling of terms works as expected

We added the following conjecture to SUMO:

*instance(p,judicialProcess)*

SUMO axiom:  $\forall x \text{ instance}(x, \text{judicialProcess}) \Rightarrow \exists y \text{ agent}(x, y)$

*agent(p,sk(p)).*

# Preliminary Experiments with SUMO



MAX-PLANCK-GESellschaft

Results:

- ⌋ Equality transformation proved scalable and useful
- ⌋ Recycling of terms works as expected

We added the following conjecture to SUMO:

*instance(p,judicialProcess)*

*agent(p,smith)*

SUMO axiom:  $\forall x \text{ instance}(x, \text{judicialProcess}) \Rightarrow \exists y \text{ agent}(x, y)$

*agent(p,smith).*

# Preliminary Experiments with SUMO



MAX-PLANCK-GESELLSCHAFT

---

## Results:

- ⌋ Equality transformation proved scalable and useful
- ⌋ Recycling of terms works as expected
- ⌋ Loop check cannot always avoid infinite models

# Conclusion

---



MAX-PLANCK-GESELLSCHAFT

Our transformation for FOL ontologies

- ↳ allows to compute models for large ontologies
- ↳ supports equality
- ↳ avoids unnecessary skolem terms
- ↳ (often) avoids infinite models