Automated Reasoning Support

for First-Order Ontologies

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

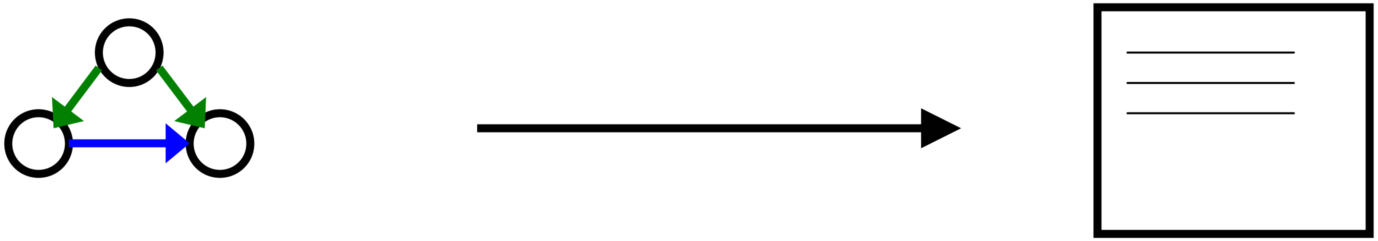
Peter Baumgartner (National ICT Australia, Canberra/Australia)

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

Automated Reasoning Support for FOLOs

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

Model Computation



Ontology
(set of FOL formulae)

Model
(set of derivable facts)

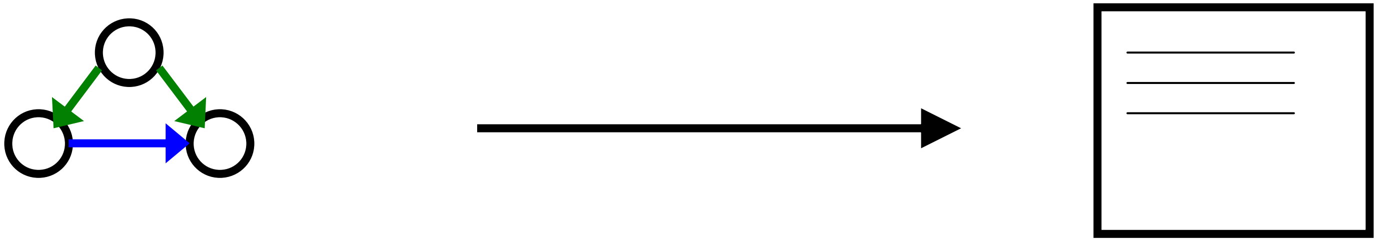
$\forall x\ singer(x) => sings(x)$

$singer(elvis)$

$sings(elvis).$

$singer(elvis).$

Use of Model Computation



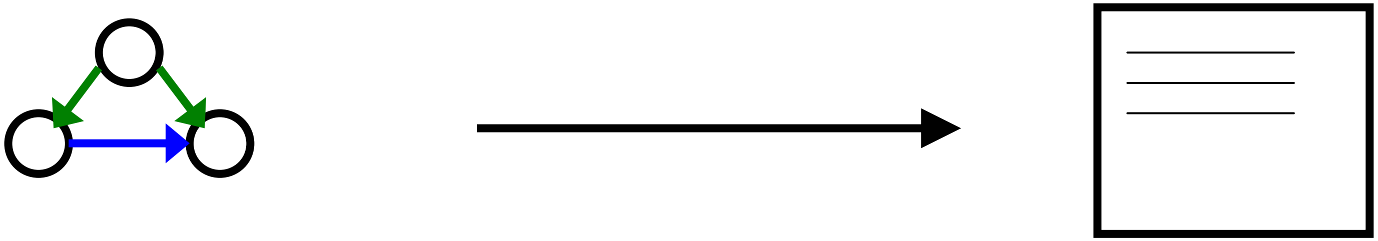
Model computation can be used for:

- Finding contradictions in the ontology

  (there is a model only iff the ontology is consistent)

- Debugging the ontology
- Proving/disproving conjectures

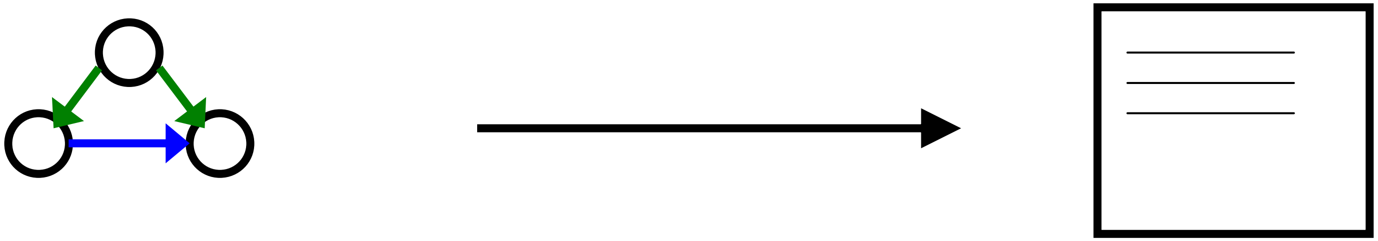
Undecidability of Model Computation



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:

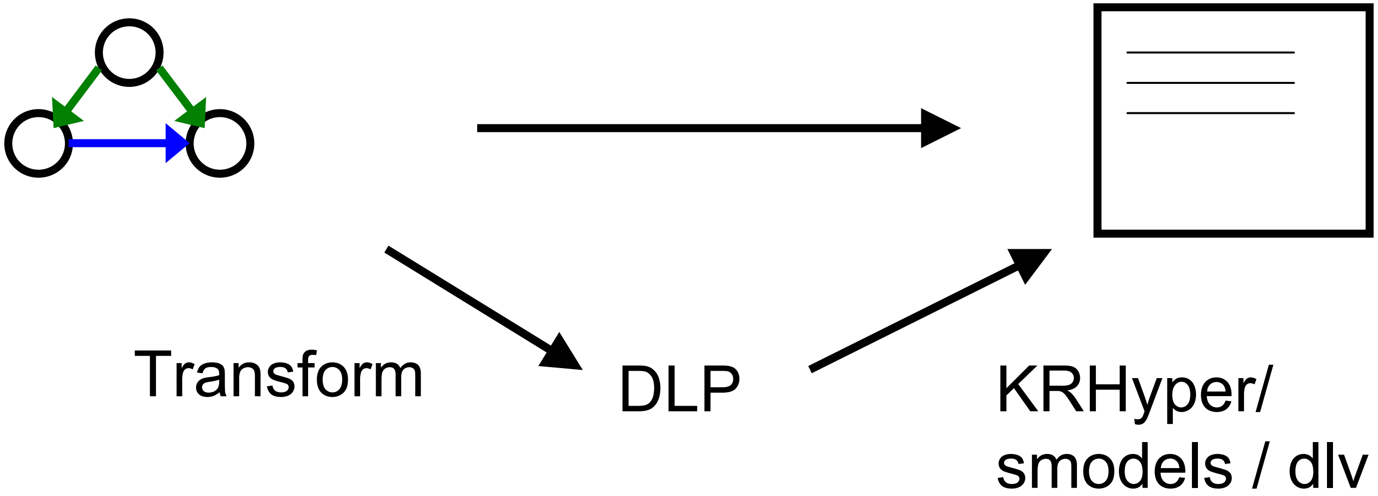
- They try to map all constants to the same domain element
- They have difficulties if there are many distinct constants

$p(c_1,\ldots c_n).$

$\neg p(x_1,\ldots,x_{i-1}, x, x_{i+1}\ldots,x_{j-1}, x, x_{j+1},\ldots x_n).$ *for all* $1<i<j<n$

Fails for n>8

Our Approach



Transform

DLP

KRHyper/
smodels / dlv

Disjunctive Logic Programs (DLPs)

Rule:

r(a) v p(x,y) :- q(x,y), r(z), not(s(a,x)).

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

Our Transformation from FOL to DLPs

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

$\Big\downarrow$ Prenex Negation Normal Form

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

Our Transformation from FOL to DLPs

$$\forall x\ inCharge(x) \Rightarrow onLeave(x) \lor \exists y\ refersTo(x,y)$$

$$\downarrow$$

$$\forall x[\dots]\ \exists y[\dots]\ [Qz\dots]\ \neg inCharge(x) \lor onLeave(x) \lor refersTo(x,y)$$

Our Transformation from FOL to DLPs

$$\forall x\ inCharge(x) \Rightarrow onLeave(x) \lor \exists y\ refersTo(x,y)$$

$$\downarrow$$

$$\forall x[\ldots]\ \exists y[\ldots]\ [Qz\ldots]\ \neg inCharge(x) \lor onLeave(x) \lor refersTo(x,y)$$

Disjuncts without existential or following variables

Disjuncts with existential or following variables

Our Transformation from FOL to DLPs

$$\forall x\ inCharge(x) \Rightarrow onLeave(x) \lor \exists y\ refersTo(x,y)$$

$$\downarrow$$

$$\forall x[\ldots]\ \exists y[\ldots]\ [Qz\ldots]\ \neg inCharge(x) \lor onLeave(x) \lor refersTo(x,y)$$

Disjuncts without existential or following variables

Disjuncts with existential or following variables

Our Transformation from FOL to DLPs

$$\forall x\ inCharge(x) => onLeave(x) \lor \exists y\ refersTo(x,y)$$

$$\downarrow$$

negative literals

$$\forall x[\ldots]\ \exists y[\ldots]\ [Qz\ldots]\ \neg inCharge(x) \lor onLeave(x) \lor refersTo(x,y)$$

Disjuncts without
existential or
following variables

Disjuncts with
existential or
following variables

Our Transformation from FOL to DLPs

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

$$\Big\downarrow$$

Option 1: Usual Skolemization

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

Our Transformation from FOL to DLPs

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

$$\Big\downarrow$$

Option 1: Usual Skolemization

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

Problem:

$\neg onLeave(Smith)$

$refersTo(Smith,Miller)$

→ refersTo(Smith,sk(Smith)).

Our Transformation from FOL to DLPs

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

$\downarrow$ Option 2: Recyling

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

Problem:

$\neg onLeave(Smith)$

$refersTo(Smith,Miller)$

→ refersTo(Smith,sk(Smith)).

(simplified. See paper)

Our Transformation from FOL to DLPs

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

$$\downarrow$$

Option 2: Recyling

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

false :- is_sat$_{\exists}$(x),
not(refersTo(x,y)).

Problem:

$\neg onLeave(Smith)$

$refersTo(Smith,Miller)$

→ refersTo(Smith,sk(Smith)).

(simplified. See paper)

Our Transformation from FOL to DLPs

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

$\downarrow$ Option 2: Recyling

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

false :- $is\_sat_{\exists}(x)$,

not(refersTo(x,y)).

Problem:

¬onLeave(Smith)

refersTo(Smith,Miller)

→ ~~refersTo(Smith,sk(Smith)).~~

(simplified. See paper)

Our Transformation from FOL to DLPs

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

$\downarrow$ Option 2: Recyling

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

false :- is_sat$_{\exists}$(x),
not(refersTo(x,y)).

Problem:

$\neg\exists x\ onLeave(x)$

$\longrightarrow$ refersTo(Smith,sk(Smith)).

Our Transformation from FOL to DLPs

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

↓ Option 2: Recyling

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

false :- $is\_sat_{\exists}(x)$,

not(refersTo(x,y)).

Problem:

$\neg\exists x\ onLeave(x)$

→ 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))))))))).

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

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

Option 2: Recyling

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

is_sat$_\exists$(x) ∨ onLeave(x) ∨ refersTo(x,sk(x)) :- inCharge(x).

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

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

not(refersTo(x,y)).

Problem:

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

→ refersTo(Smith,sk(Smith)).

sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(sk

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

sk(sk(sk(sk(sk(sk(sk(sk(sk(sk(s

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

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

Our Transformation from FOL to DLPs

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

$$\Big\downarrow$$

Option 2: Recyling

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

$$\text{false :- is\_sat}_{\exists}(x),$$
$$\text{not(refersTo}(x,y)).$$

Problem:

$\neg \exists x \; onLeave(x)$

→ refersTo(Smith,sk(Smith)).

(simplified. See paper)

Our Transformation from FOL to DLPs

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

$$\big\downarrow$$

Option 3: Loop check

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

Problem:

$\neg\exists x\ onLeave(x)$

→ refersTo(Smith,sk(Smith)).

(simplified. See paper)

Our Transformation from FOL to DLPs

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

$\downarrow$ Option 3: Loop check

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

false :- is_sat$_{\exists}$(x),

not(refersTo(x,y)).

refersTo(x,sk(z)) :-

prev$_{\exists}$(x),

refersTo(z,sk(z)).

Problem:

$\neg\exists x\ onLeave(x)$

→ refersTo(Smith,sk(Smith)).

(simplified. See paper)

Our Transformation from FOL to DLPs

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

↓ Option 3: Loop check

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

false :- is_sat$_\exists$(x),

not(refersTo(x,y)).

refersTo(x,sk(z)) :-

prev$_\exists$(x),

refersTo(z,sk(z)).

Problem:

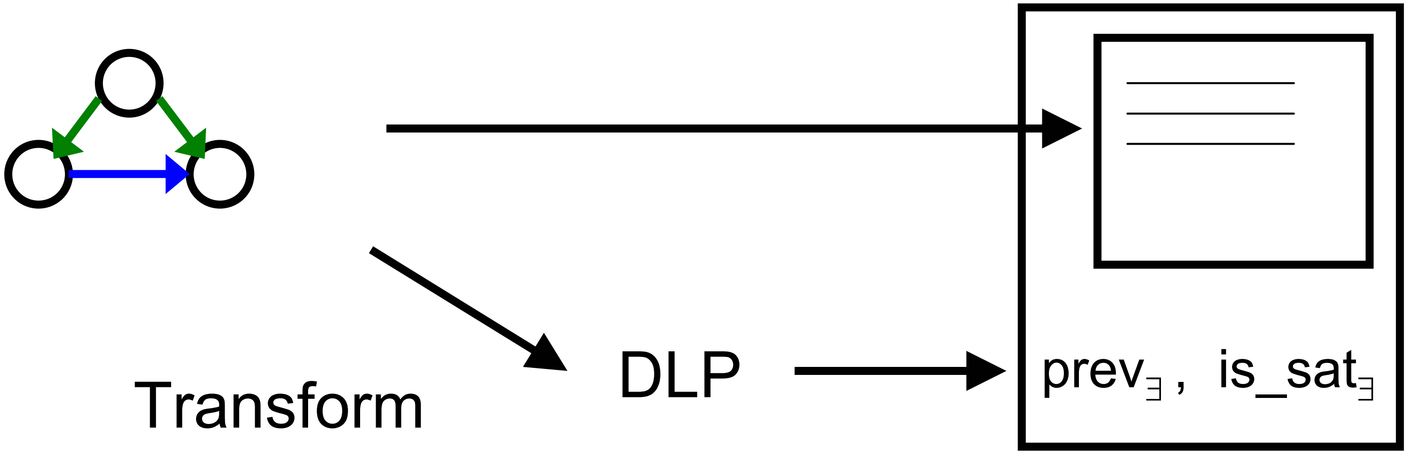
$\neg\exists x\ onLeave(x)$

→ refersTo(Smith,sk(Smith)).

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

(simplified. See paper)

Our Transformation from FOL to DLPs



Transform

DLP

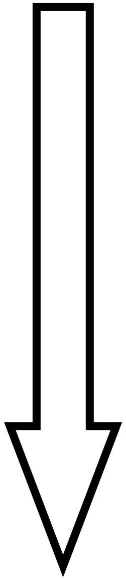
$\text{prev}_\exists$, $\text{is\_sat}_\exists$

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

Treating Equality

equal(dog(smith), bonzo).

inCharge(dog(smith)).



inCharge(bonzo).

Treating Equality

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

inCharge(dog(smith)).
```

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

Substitution axioms

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

Equivalence axioms

Treating Equality

equal(dog(smith), bonzo).

inCharge(dog(smith)).

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

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

Substitution axioms

equal(X,X).

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

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

Equivalence axioms

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

Treating Equality

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

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

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

```
equal(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(elvi
equal(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(smith)))))
equal(dog(dog(dog(dog(dog(dog(dog(dog(dog(smith))))))))),w
equal(dog(dog(dog(dog(dog(dog(dog(dog(smith)))))))),dog(wi
equal(dog(dog(dog(dog(dog(dog(dog(smith))))))),dog(dog(wif
equal(dog(dog(dog(dog(dog(dog(smith)))))),dog(dog(dog(wif
equal(dog(dog(dog(dog(dog(smith))))),dog(dog(dog(dog(pris
equal(dog(dog(dog(dog(smith)))),dog(dog(dog(bonzo)))).
equal(dog(dog(dog(smith))),dog(dog(bonzo))).
equal(dog(dog(smith)),dog(bonzo)).
```

Treating Equality

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

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

Substitution axioms

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

Equivalence axioms

Treating Equality

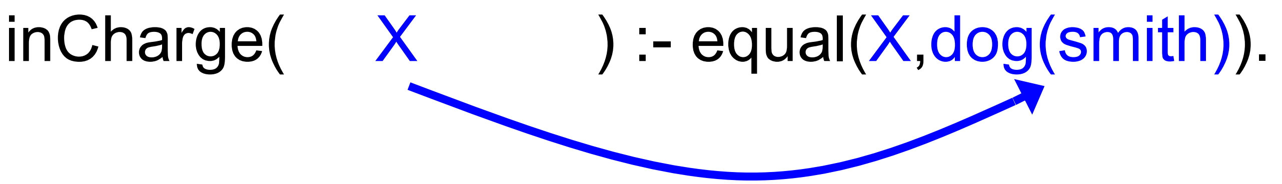
```
equal(dog(smith), bonzo).
inCharge(dog(smith)).
```

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

Equivalence axioms

Treating Equality

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

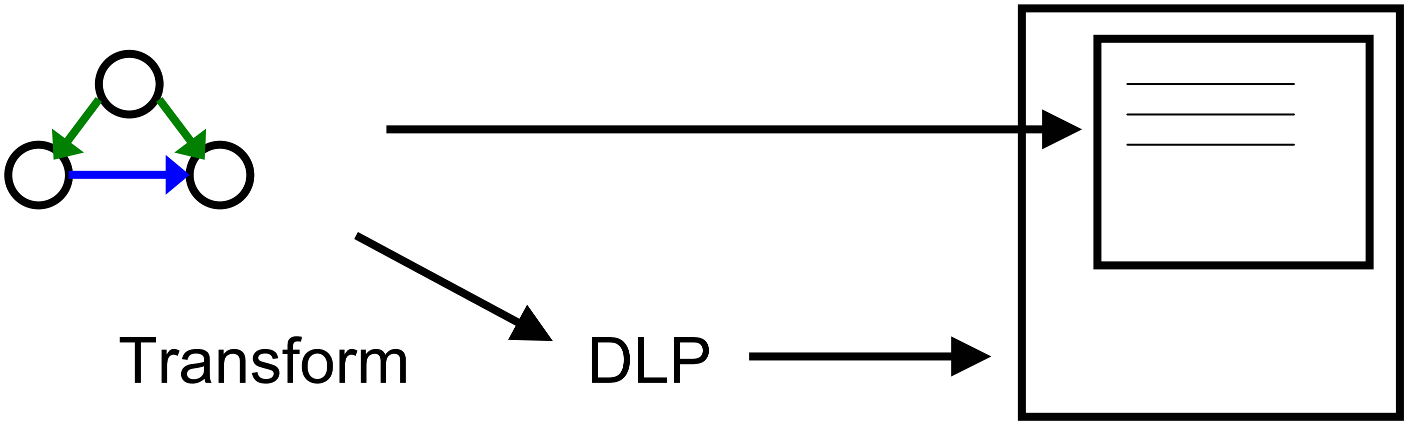
```
equal(dog(smith), bonzo).
inCharge(   X        ) :- equal(X,dog(smith)).
```

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

Equivalence axioms

```
inCharge(bonzo).
```

Treating Equality

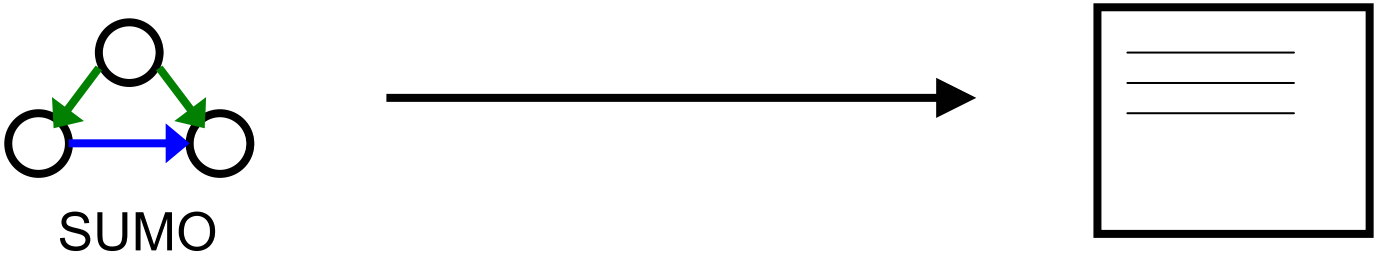


Transform DLP

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



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

$$\forall x\ human(x) => \exists y\ mother(x,y)$$

Preliminary Experiments with SUMO

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

Results:

- Equality transformation proved scalable and useful

Preliminary Experiments with SUMO

Results:

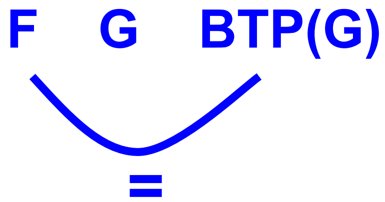
- Equality transformation proved scalable and useful

  We added the following conjectures to SUMO:

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

  orientation(france, west, germany).

  equal(biggestTradingPartner(germany),france).



F G BTP(G)

=

Preliminary Experiments with SUMO

Results:

- Equality transformation proved scalable and useful

  We added the following conjectures to SUMO:

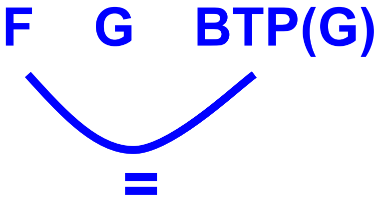
orientation(germany,west, biggestTradingPartner(germany)).

orientation(france, west, germany).

equal(biggestTradingPartner(germany),france).

orientation(germany, east, france).

orientation(germany, west, france). ⚡



Preliminary Experiments with SUMO

Results:

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

Preliminary Experiments with SUMO

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

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: $\forall x\ instance(x,judicialProcess) \Rightarrow \exists y\ agent(x,y)$

agent(p,sk(p)).

Preliminary Experiments with SUMO

Results:

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

We added the following conjecture to SUMO:

$\mathit{instance}(p,\mathit{judicialProcess})$

$\mathit{agent}(p,\mathit{smith})$

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

agent(p,smith).

Preliminary Experiments with SUMO

Results:

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

Conclusion

Our transformation for FOL ontologies

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