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 FOL Ontologies

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

Ontology
(set of FOL formulae)

\[ \forall x \, \text{singer}(x) \Rightarrow \text{sings}(x) \]
\[ \text{singer}(\text{elvis}) \]

Model
(set of derivable facts)

\[ \text{sings}(\text{elvis}). \]
\[ \text{singer}(\text{elvis}). \]
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

\[
p(c_1, \ldots, c_n).
\]

\[
\neg p(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_{j-1}, x, x_{j+1}, \ldots, x_n) \quad \text{for all } 1 < i < j < n
\]

Fails for \( n > 8 \)
Our Approach

Transform → DLP → KRHyper/ smodels / dlv
Disjunctive Logic Programs (DLPs)

Rule:

\[
\text{r(a) } \lor \text{ p(x,y) } :- \text{ q(x,y), r(z), not(s(a,x))}. 
\]

A Disjunctive Logic Program (DLP) is a set of rules.
Our Transformation from FOL to DLPs

\[ \forall x \text{ inCharge}(x) \Rightarrow \text{onLeave}(x) \lor \exists y \text{ refersTo}(x,y) \]

Prenex Negation Normal Form

\[ \forall x[...] \exists y[...] [Qz...] \neg\text{inCharge}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x,y) \]
Our Transformation from FOL to DLPs

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

\[ \forall x[\ldots] \exists y[\ldots] \ [Qz\ldots] \ \neg inCharge(x) \lor onLeave(x) \lor refersTo(x, y) \]
Our Transformation from FOL to DLPs

\[ \forall x \text{ inCharge}(x) \Rightarrow \text{onLeave}(x) \lor \exists y \text{ refersTo}(x, y) \]

\[ \forall x \ldots \exists y \ldots \left[ Qz \ldots \right] \neg \text{inCharge}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x, y) \]

- Disjuncts without existential or following variables
- Disjuncts with existential or following variables
Our Transformation from FOL to DLPs

∀x inCharge(x) ⇒ onLeave(x) ∨ ∃y refersTo(x,y)

∀x[...] ∃y[...] [Qz...] ¬inCharge(x) ∨ onLeave(x) ∨ refersTo(x,y)

Disjuncts without existential or following variables
Disjuncts with existential or following variables
Our Transformation from FOL to DLPs

\[ \forall x \text{ inCharge}(x) \Rightarrow \text{onLeave}(x) \lor \exists y \text{ refersTo}(x, y) \]

\[ \forall x[...] \exists y[...] [Qz...] \neg \text{inCharge}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x, y) \]

Disjuncts without existential or following variables

Disjuncts with existential or following variables
Our Transformation from FOL to DLPs

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

Option 1: Usual Skolemization

\[ onLeave(x) \lor refersTo(x,sk(x)) \ :- \ inCharge(x). \]
Our Transformation from FOL to DLPs

\[ \forall x \exists y \, \neg \text{inCharge}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x, y) \]

Option 1: Usual Skolemization

\[ \text{onLeave}(x) \lor \text{refersTo}(x, \text{sk}(x)) \implies \text{inCharge}(x). \]

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

\[ \forall x \exists y \; \neg \text{inCharge}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x,y) \]

Option 2: Recycling

\[ \text{is}_{	ext{sat}_3}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x,\text{sk}(x)) \leftarrow \text{inCharge}(x). \]

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

∀x ∃y \neg inCharge(x) ∨ onLeave(x) ∨ refersTo(x,y)

Option 2: Recyling

is_sat₃(x) ∨ onLeave(x) ∨ refersTo(x,sk(x)) :- inCharge(x).

false :- is_sat₃(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

\[ \forall x \exists y \neg \text{inCharge}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x,y) \]

Option 2: Recycling

\[ \text{is_sat}_3(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x,\text{sk}(x)) :\!- \text{inCharge}(x). \]

false :\!- \text{is_sat}_3(x),

\[ \text{not}(\text{refersTo}(x,y)). \]

Problem:

\[ \neg \text{onLeave}(Smith) \]

\[ \text{refersTo}(Smith, Miller) \]

\[ \text{refersTo}(Smith, \text{sk}(Smith)). \]

(simplified. See paper)
Our Transformation from FOL to DLPs

\[ \forall x \exists y \neg \text{inCharge}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x,y) \]

Option 2: Recyling

\[ \text{is}_\exists \text{sat}_3(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x,\text{sk}(x)) :- \text{inCharge}(x). \]

false :- \text{is}_\exists \text{sat}_3(x),

not(\text{refersTo}(x,y)).

Problem:

\[ \neg \exists x \text{onLeave}(x) \]

\[ \Rightarrow \text{refersTo}(\text{Smith},\text{sk}(\text{Smith})). \]
Our Transformation from FOL to DLPs

\[ \forall x \, \exists y \, \neg \text{inCharge}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x, y) \]

Option 2: Recycling

\[ \exists x \, \text{is\_sat}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x, \text{sk}(x)) \iff \text{inCharge}(x). \]

false \iff \exists x \, \text{is\_sat}(x),
\neg \text{refersTo}(x, y)).

Problem:

\[ \exists x \, \text{onLeave}(x) \]
\[ \rightarrow \text{refersTo}(\text{Smith}, \text{sk}(\text{Smith})). \]
\[ \text{refersTo}(\text{sk}(\text{Smith}), \text{sk}(\text{sk}(\text{Smith}))). \]
\[ \text{refersTo}(\text{sk}(\text{sk}(\text{Smith})), \text{sk}(\text{sk}(\text{sk}(\text{Smith}))).) \]
\[ \text{refersTo}(\text{sk}(\text{sk}(\text{sk}(\text{Smith}))), \text{sk}(\text{sk}(\text{sk}(\text{sk}(\text{Smith}))))). \]
Our Transformation from FOL to DLPs

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

Option 2: Recycling

\[ \text{is_sat}(x) \vee \text{onLeave}(x) \vee \text{refersTo}(x, sk(x)) \leftarrow \text{inCharge}(x) \]

Problem:

\[ \neg \exists x \text{onLeave}(x) \]

false \leftarrow \text{is_sat}(x),

not(\text{refersTo}(x, y)).
Our Transformation from FOL to DLPs

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

Option 2: Recyling

\[ is_{\text{sat}_3}(x) \lor onLeave(x) \lor refersTo(x,sk(x)) :- inCharge(x). \]

false :- \( is_{\text{sat}_3}(x), \)
\[ \text{not}(refersTo(x,y)). \]

Problem:

\[ \neg \exists x onLeave(x) \]\n\[ \rightarrow refersTo(Smith,sk(Smith)). \]

(simplified. See paper)
Our Transformation from FOL to DLPs

\[ \forall x \exists y \neg \text{inCharge}(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x,y) \]

Option 3: Loop check

\[ \text{prev}_\exists(x) \lor \text{is}_{\text{sat}}_\exists(x) \lor \text{onLeave}(x) \lor \text{refersTo}(x,\text{sk}(x)) \implies \text{inCharge}(x). \]

Problem:

\[ \neg \exists x \text{onLeave}(x) \implies \text{refersTo}(\text{Smith},\text{sk}(\text{Smith})). \]

(simplified. See paper)
Our Transformation from FOL to DLPs

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

Option 3: Loop check

prev_∃(x) \lor is_sat_∃(x) \lor onLeave(x) \lor refersTo(x,sk(x)) :- inCharge(x).

false :- is_sat_∃(x),

\quad not(refersTo(x,y)).

refersTo(x,sk(z)) :- prev_∃(x),

\quad refersTo(z,sk(z)).

Problem:

\[ \rightarrow \exists x onLeave(x) \]

\[ \rightarrow refersTo(Smith,sk(Smith)). \]

(simplified. See paper)
Our Transformation from FOL to DLPs

∀x ∃y ¬inCharge(x) ∨ onLeave(x) ∨ refersTo(x,y)

Option 3: Loop check

prev∃(x) ∨ is_sat∃(x) ∨ onLeave(x) ∨ refersTo(x,sk(x)) :- inCharge(x).

false :- is_sat∃(x),
not(refersTo(x,y)).

refersTo(x,sk(z)) :-
prev∃(x),
refersTo(z,sk(z)).

Problem:

¬∃x onLeave(x)

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

(simplified. See paper)
Our Transformation from FOL to DLPs

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

equal(dog(smith), bonzo).

inCharge(dog(smith)).
Treating Equality

equal(dog(smith), bonzo).

inCharge(dog(smith)).

\[
\begin{align*}
inCharge(Y) & : \text{-} inCharge(X), equal(X, Y) \\
equal(dog(X), dog(Y)) & : \text{-} equal(X, Y) \\
equal(X, X) & \\
equal(X, Y) & : \text{-} equal(Y, X) \\
equal(X, Z) & : \text{-} equal(X, Y), equal(Y, Z).
\end{align*}
\]

Substitution axioms

Equivalence axioms
Treating Equality

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

\[
\begin{align*}
\text{inCharge}(Y) & : \text{inCharge}(X), \text{equal}(X,Y) \\
equal(\text{dog}(X), \text{dog}(Y)) & : \text{equal}(X,Y) \\
equal(X,X) & \\
equal(X,Y) & : \text{equal}(Y,X) \\
equal(X,Z) & : \text{equal}(X,Y), \text{equal}(Y,Z).
\end{align*}
\]

\[
equal(\text{dog}(\text{dog}(\text{smith})), \text{dog}(\text{bonzo})).
\]
Treating Equality

\begin{align*}
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(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(smith))))))))))),w\\
equal&(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(dog(smith))))))))))),dog(wif\\
equal&(dog(dog(dog(dog(dog(dog(dog(dog(dog(smith))))))))),dog(dog(wif\\
equal&(dog(dog(dog(dog(dog(dog(dog(dog(smith))))))))),dog(dog(dog(wif\\
equal&(dog(dog(dog(dog(dog(dog(dog(dog(smith))))))))),dog(dog(dog(dog(smith)))))))))\text{,dog(dog(dog(dog(bonzo)))).}
\end{align*}

Substitution axioms

\begin{align*}
in\text{Charge}(Y):-in\text{Charge}(X),equal(X,Y) \\
equal&(dog(X),dog(Y)) :- equal(X,Y).
\end{align*}

Equivalence axioms

\begin{align*}
equal&(X,X). \\
equal&(X,Y) :- equal(Y,X). \\
equal&(X,Z) :- equal(X,Y), equal(Y,Z).
\end{align*}
Treating Equality

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

\[
\begin{align*}
inCharge(Y) & \leftarrow \text{inCharge}(X), \text{equal}(X,Y), \\
equal(dog(X), dog(Y)) & \leftarrow \text{equal}(X,Y). \\
equal(X, X) & \\
equal(X, Y) & \leftarrow \text{equal}(Y, X). \\
equal(X, Z) & \leftarrow \text{equal}(X, Y), \text{equal}(Y, Z).
\end{align*}
\]

Substitution axioms
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).

inCharge(bonzo).
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

$$\forall x \ human(x) \Rightarrow \exists y \ mother(x,y)$$
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:

  \[ \text{orientation(germany, west, biggestTradingPartner(germany)).} \]

  \[ \text{orientation(france, west, germany).} \]

  \[ \text{equal(biggestTradingPartner(germany), france).} \]
Results:

Equality transformation proved scalable and useful
We added the following conjectures to SUMO:

\[
\begin{align*}
\text{orientation}(\text{germany}, \text{west}, \text{biggestTradingPartner}(\text{germany})). \\
\text{orientation}(\text{france}, \text{west}, \text{germany}). \\
\text{equal}(\text{biggestTradingPartner}(\text{germany}), \text{france}). \\
\text{orientation}(\text{germany}, \text{east}, \text{france}). \\
\text{orientation}(\text{germany}, \text{west}, \text{france}).
\end{align*}
\]
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:

\[ \text{instance}(p, \text{judicialProcess}) \]
Results:

- Equality transformation proved scalable and useful
- Recycling of terms works as expected
  
  We added the following conjecture to SUMO:

  \[ \text{instance}(p, \text{judicialProcess}) \]

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

  \[ \text{agent}(p, \text{sk}(p)). \]
Preliminary Experiments with SUMO

Results:

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

We added the following conjecture to SUMO:

\[
\begin{align*}
\text{instance}(p, \text{judicialProcess}) \\
\text{agent}(p, \text{smith}) \\
\text{SUMO axiom: } \forall x \text{ instance}(x, \text{judicialProcess}) &\Rightarrow \exists y \text{ agent}(x,y) \\
\text{agent}(p, \text{smith}).
\end{align*}
\]
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