Verifying the Equivalence of Logic Programs in the Disjunctive Case

Emilia Oikarinen and Tomi Janhunen
Helsinki University of Technology
Laboratory for Theoretical Computer Science
{emilia.oikarinen,tomi.janhunen}@hut.fi

E. Oikarinen and T. Janhunen — Verifying the Equivalence of Logic Programs in the Disjunctive Case

Motivation

- Solving a problem in answer set programming (ASP) typically results in several versions of the logic program formalizing the problem.
- Problem: how to ensure that different encodings yield the same output i.e. have the same answer sets?
- We consider the following two notions of equivalence
  - Logic programs P and Q are *(weakly) equivalent* ($P \equiv Q$) $\iff$ P and Q have exactly the same answer sets.
  - Logic programs P and Q are *strongly equivalent* ($P \equiv_{\mathrm{s}} Q$) $\iff$ $P \cup R \equiv Q \cup R$ for all logic programs R.

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

E. Oikarinen and T. Janhunen — Verifying the Equivalence of Logic Programs in the Disjunctive Case

Outline

- Motivation: Equivalence of Logic Programs
- Disjunctive Logic Programs: Syntax and Semantics
- Translation-based Verification Method
- Experiments
- Conclusions

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

E. Oikarinen and T. Janhunen — Verifying the Equivalence of Logic Programs in the Disjunctive Case

Motivation Cont'd

- We consider *(weak) equivalence* of *disjunctive logic programs*.
- We have previously developed an automated translation-based method for verifying the equivalence of programs supported by the SMODELS system.
- $P \equiv_{\mathrm{s}} Q \implies P \equiv Q$ (by setting $R = \emptyset$), but $P \equiv Q \not\Longrightarrow P \equiv_{\mathrm{s}} Q$.
- Whether $P \equiv Q$ holds, remains open whenever $P \not\equiv_{\mathrm{s}} Q$ holds $\implies$ Verifying $P \equiv Q$ remains as a problem of its own.
- Complexity results support this view: deciding $P \equiv Q$ for finite propositional disjunctive programs is $\Pi_2^{\mathbf{P}}$-hard whereas deciding $P \equiv_{\mathrm{s}} Q$ is only $\mathbf{coNP}$-complete.

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

Disjunctive Logic Programs

- A (propositional) *disjunctive logic program* (DLP) P is a set of *rules* of the form

$$a_1 \mid \ldots \mid a_n \leftarrow b_1, \ldots, b_m, {\sim}c_1, \ldots, {\sim}c_k,$$

  where $a_1, \ldots, a_n, b_1, \ldots, b_m, c_1, \ldots, c_k$ are propositional atoms and n, k, m are natural numbers.
- A shorthand: $A \leftarrow B, {\sim}C$.
- Program P is *normal* if $n = 1$ for each rule of P.
- Program P is *positive* if $k = 0$ for each rule of P.

E. Oikarinen and T. Janhunen Verifying the Equivalence of Logic Programs in the Disjunctive Case

Stable Model Semantics

- Given a DLP P and $M \subseteq \mathrm{Hb}(P)$, the *Gelfond-Lifschitz reduct* of P is a positive program

$$P_M = \{A \leftarrow B \mid A \leftarrow B, {\sim}C \in P \text{ and } M \cap C = \emptyset\}.$$

- M is a *stable model* of $P \iff M \in \mathbf{MM}(P_M)$.
- We denote the set of stable models of P by $\mathbf{SM}(P)$.

Example. Consider $P = \{a \mid b \leftarrow {\sim}b.\ b \leftarrow {\sim}a\}$ and $M = \{a\}$. Now, $P_M = \{a \mid b \leftarrow\ \}$ and $\mathbf{MM}(P_M) = \{\{a\}, \{b\}\}$. Thus $M \in \mathbf{SM}(P)$.

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

E. Oikarinen and T. Janhunen Verifying the Equivalence of Logic Programs in the Disjunctive Case

Satisfaction Relation and Minimal Models

- The *Herbrand base* $\mathrm{Hb}(P)$ is the set of atoms appearing in P.
- An *interpretation* $I \subseteq \mathrm{Hb}(P)$ of P defines which atoms $a \in \mathrm{Hb}(P)$ are true ($a \in I$) and which are false ($a \notin I$).
- An interpretation I is a (classical) *model* of P ($I \models P$) $\iff$ for each $A \leftarrow B, {\sim}C \in P$, $B \subseteq I$ and $C \cap I = \emptyset$ imply $A \cap I \neq \emptyset$.
- M is a *minimal model* of P, if there is no $M' \subset M$ such that $M' \models P$. The set of minimal models of P is denoted by $\mathbf{MM}(P)$.

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

E. Oikarinen and T. Janhunen Verifying the Equivalence of Logic Programs in the Disjunctive Case

Verifying Equivalence

- We assume that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ without loss of generality, since $\{a \leftarrow a\} \equiv_s \emptyset$.
- We consider a translation $\mathrm{TR}(P, Q)$ such that $\mathrm{TR}(P, Q)$ has a stable model $\iff \exists M \in \mathbf{SM}(P)$ s.t. $M \notin \mathbf{SM}(Q)$. Thus,

$$P \equiv Q \iff \mathbf{SM}(\mathrm{TR}(P, Q)) = \emptyset \text{ and } \mathbf{SM}(\mathrm{TR}(Q, P)) = \emptyset.$$

- We can distinguish two types of counter-examples for equivalence.
  - T1: $\langle M, M \rangle$ s.t. $M \in \mathbf{SM}(P)$ and $M \not\models Q_M$.
  - T2: $\langle M, M' \rangle$ s.t. $M \in \mathbf{SM}(P)$, $M \models Q_M$, $M' \subset M$ and $M' \models Q_M$.

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science

The translation $\mathrm{TR}(P,Q)$ contains

- all the rules of P without modifications,
- a rule $unsat \leftarrow B, \sim(A \cup C)$ for each rule $A \leftarrow B, \sim C \in Q$,
- rules $a^\bullet \leftarrow a, \sim a^\circ, \sim unsat$ and $a^\circ \leftarrow a, \sim a^\bullet, \sim unsat$ for each atom $a \in \mathrm{Hb}(P)$,
- a rule $unsat^\bullet \leftarrow B^\bullet, \sim(A^\bullet \cup C), \sim unsat$ for each rule $A \leftarrow B, \sim C \in Q$,
- a rule $\mathit{diff} \leftarrow a, \sim a^\bullet, \sim unsat$ for each atom $a \in \mathrm{Hb}(P)$ and
- rules $ok \leftarrow unsat$; $ok \leftarrow \mathit{diff}, \sim unsat, \sim unsat^\bullet$ and $\perp \leftarrow \sim ok$.

Two-Phased Translation

- Since there are two types of counter-examples for equivalence, testing can be performed in two phases.
  - *Phase 1*: $\mathbf{SM}(\mathrm{TR}_1(P,Q)) \neq \emptyset$
    $\iff \exists M \in \mathbf{SM}(P)$ s.t. $M \not\models Q_M$,
    i.e. there exists a counter-example of type T1.
  - *Phase 2* (if $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$): $\mathbf{SM}(\mathrm{TR}_2(P,Q)) \neq \emptyset$
    $\iff \exists M \in \mathbf{SM}(P)$ s.t. $M \notin \mathbf{MM}(Q_M)$,
    i.e. there exists a counter-example of type T2.
- $\mathrm{TR}_1(P,Q)$ and $\mathrm{TR}_2(P,Q)$ can easily be obtained from $\mathrm{TR}(P,Q)$.

Example

- Programs $P = \{a \mid b\}$ and $Q = \{a \leftarrow \sim b\}$. The translation

$$\begin{aligned}\mathrm{TR}(P,Q) = \{&a \mid b.\ unsat \leftarrow \sim a, \sim b.\ unsat^\bullet \leftarrow \sim a^\bullet, \sim b, \sim unsat\\ &a^\bullet \leftarrow a, \sim a^\circ, \sim unsat.\ a^\circ \leftarrow a, \sim a^\bullet, \sim unsat\\ &b^\bullet \leftarrow b, \sim b^\circ, \sim unsat.\ b^\circ \leftarrow b, \sim b^\bullet, \sim unsat\\ &\mathit{diff} \leftarrow a, \sim a^\bullet, \sim unsat.\ \mathit{diff} \leftarrow b, \sim b^\bullet, \sim unsat\\ &ok \leftarrow unsat.\ ok \leftarrow \mathit{diff}, \sim unsat, \sim unsat^\bullet.\ \perp \leftarrow \sim ok\}.\end{aligned}$$

- Consider interpretation $N = \{b, b^\circ, \mathit{diff}, ok\}$:

$$\begin{aligned}\mathrm{TR}(P,Q)_N = \{&a \mid b.\ a^\bullet \leftarrow a.\ a^\circ \leftarrow a.\ b^\circ \leftarrow b.\ \mathit{diff} \leftarrow a.\ \mathit{diff} \leftarrow b\\ &ok \leftarrow unsat.\ ok \leftarrow \mathit{diff}\}.\end{aligned}$$

- $\mathbf{MM}(\mathrm{TR}(P,Q)_N) = \{\{a, a^\bullet, a^\circ, \mathit{diff}, ok\}, \{b, b^\circ, \mathit{diff}, ok\}\}$, thus $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$, $\mathbf{SM}(\mathrm{TR}(P,Q)) \neq \emptyset$ and therefore $P \not\equiv Q$.

Experiments

- The translation functions have been implemented in C under Linux and a *naive cross-checking approach* as a shell script.
- The current implementation DLPEQ is available in the web:

  `http://www.tcs.hut.fi/Software/lpeq/`

- The performance of the naive and the two translation-based approaches was compared in several experiments.
- A two-way search of counter-examples was performed in any case.
- GNT was used for the computation of stable models.

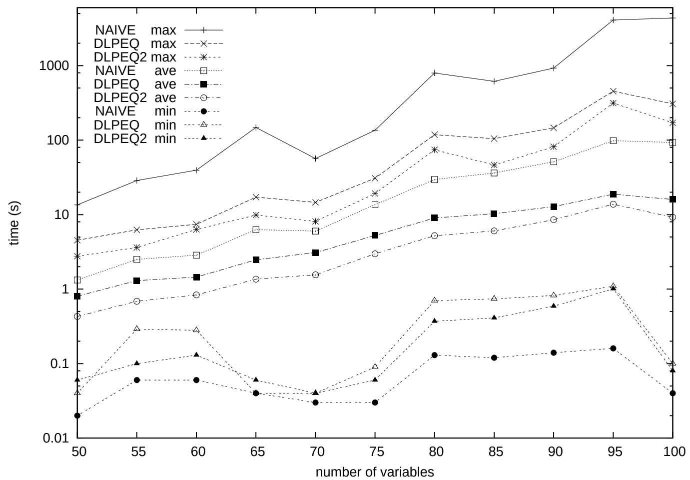
Disjunctive Random 3-SAT

- We use problem of finding a minimal model of a random 3-SAT instance containing specified atoms as our first test problem.
- Encoding as DLPs that solve an instance of a random 3-SAT problem and additional rules for random atoms c_i, for $i = 1, \ldots, \lfloor 2v/100 \rfloor$, where v is the number of atoms.
- A fixed clauses to variables ratio $c/v = 3.5$.
- We test the equivalence of each program P against a variant P' obtained by dropping a random rule from P.

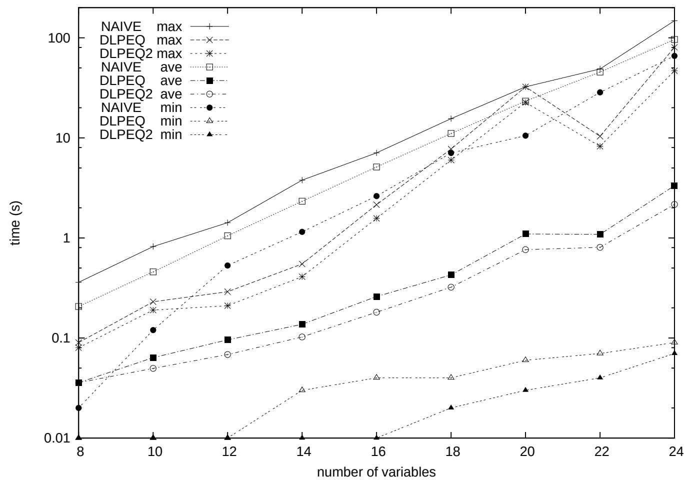
Random 2-QBF

- Similarly to the previous experiment, but using DLP encodings of random 2-QBF instances $\Phi = \exists X \forall Y \phi$, where ϕ is a 3-SAT instance in DNF over $X \cup Y$.
- $|X| = |Y|$, $v = |X| + |Y|$ and $c/v = 3.5 =$ constant.

Results: Disjunctive Random 3-SAT



Results: Random 2-QBF



Conclusions

- Two translation-based methods and an implementation for verifying the equivalence of DLPs have been presented.
- In many cases, the time needed for computations is less than in a naive approach of cross-checking the stable models.
- If programs have no/few stable models, then the naive approach can become superior to the translation-based ones.
- Two-phased translation is faster than the one-phased one.
- **Future work:** experiments using real-life problems, extension to other classes of logic programs, other notions of equivalence.