Lecture 9: Equivalence Checking

Outline
1. Motivation
2. Notions of equivalence
3. Complexity analysis
4. Translation-based verification
5. Tool for equivalence testing
6. Experimental results

1. MOTIVATION

- The program development in ASP resembles that in conventional programming languages: the final program solving a particular problem is obtained after a number of changes to the first version.
- Sometimes the aim is to change the set of answer sets whereas some steps aim at a better performance.
- A basic question is whether the different versions of a program yield the same answer sets—corresponding to solutions.
- Logic programs $P$ and $Q$ are considered to be (weakly) equivalent, denoted by $P \equiv Q$, if and only if $SM(P) = SM(Q)$.
- We are mainly interested in the verification of $P \equiv Q$ for programs $P$ and $Q$ expressed in the input language of the smodels solver.

The Language of Interest

- The current smodels solver supports internally four types of propositional rules:
  1. normal/basic rules $a \leftarrow b_1, \ldots , b_n, \neg c_1, \ldots , \neg c_m$
  2. cardinality rules $a \leftarrow l \{b_1, \ldots , b_n, \neg c_1, \ldots , \neg c_m\}$ with $l \geq 0$,
  3. choice rules $\{a_1, \ldots , a_h\} \leftarrow b_1, \ldots , b_n, \neg c_1, \ldots , \neg c_m$, and
  4. weight rules

\[ a \leftarrow l [b_1 = w_1, \ldots , b_n = w_n, \neg c_1 = v_1, \ldots , \neg c_m = v_m] \]

where weights $l \geq 0$, $w_1 \geq 0, \ldots , w_n \geq 0$, and $v_1 \geq 0, \ldots , v_m \geq 0$.

- The front-end of the solver, lparse, supports an extended syntax that is translated into rules of the kinds listed above.

Review of the Stable Model Semantics

Definition. For an smodels program $P$ and an interpretation $M \subseteq Hb(P)$, the **reduct** $P^M$ contains

- a normal rule $a \leftarrow b_1, \ldots , b_n \iff$ there is a basic rule (1.) in $P$ such that $M \models \{\neg c_1, \ldots , \neg c_m\}$, or there is a choice rule (3.) in $P$ such that $a \in \{a_1, \ldots , a_h\}$, $M \models a$, and $M \models \{\neg c_1, \ldots , \neg c_m\}$.
- a cardinality rule $a \leftarrow l' \{b_1, \ldots , b_n\} \iff$ there is a cardinality rule (2.) in $P$ and $l' = \max (0, l - \{c_i | M \models \neg c_i\})$,
- a weight rule $a \leftarrow l'[b_1 = w_1, \ldots , b_n = w_n] \iff$ there is a weight rule (4.) in $P$ and $l' = \max (0, l - \sum_{i=1}^{m} v_i)$. 

Definition. An interpretation $M \subseteq Hb(P)$ is a **stable model** of $P$

\[ \iff M = LM(P^M), \text{i.e., the (unique) least model of } P^M. \]
2. NOTIONS OF EQUIVALENCE

- The basic notions of equivalence that have been proposed for logic programs are weak/ordinary equivalence and strong equivalence.
- The second equivalence relation takes the potential contexts of programs being compared into account.

**Definition.** smodels programs $P$ and $Q$ are (weakly) equivalent, denoted by $P \equiv Q$, if and only if $\text{SM}(P) = \text{SM}(Q)$.

**Definition.** smodels programs $P$ and $Q$ are strongly equivalent, denoted by $P \equiv_s Q$, if and only if for all smodels programs $R$, $P \cup R \equiv Q \cup R$, i.e., $\text{SM}(P \cup R) = \text{SM}(Q \cup R)$.

**Proposition.** For all smodels programs $P$ and $Q$, $P \equiv Q$ implies $P \equiv_s Q$, but not vice versa, and $P \cup R \equiv_s Q \cup R$ (congruence).

---

Characterization of Strong Equivalence

- Given an smodels program $P$, an SE-interpretation is a pair $\langle N, M \rangle$ of ordinary interpretations such that $N \subseteq M \subseteq \text{Hb}(P)$.
- An SE-interpretation $\langle N, M \rangle$ for $P$ is an SE-model of $P$ if and only if $M \models P$ and $N \models P^M$.

**Theorem.** For smodels programs $P$ and $Q$, it holds that $P \equiv_s Q$ if and only if $\text{SE}(P) = \text{SE}(Q)$, i.e., $P$ and $Q$ have the same SE-models.

**Example.** Consider $P = \{ a \leftarrow b. a \leftarrow \neg b. \}$ and $Q = \{ a. \}$ from the previous slide. The fact that $P \not\equiv_s Q$ is witnessed by
- the context $R = \{ b \leftarrow a. \}$, and
- an SE-model $\langle \emptyset, \{ a, b \} \rangle$ which is not an SE-model of $Q$.

Which SE-interpretations are the other SE-models of $P$ and $Q$?

---

3. COMPLEXITY ANALYSIS

- The question is whether it is computationally feasible to verify $P \equiv Q$ (or $P \equiv_s Q$) for two programs under consideration.
- To ease complexity analysis, we distinguish the respective implication problems for $\equiv$ and $\equiv_s$ as follows.

**Definition.**

1. The language WIMPL is the set of pairs $\langle P, Q \rangle$ of finite smodels programs such that $\text{SM}(P) \subseteq \text{SM}(Q)$.
2. The language SIMPL is the set of pairs $\langle P, Q \rangle$ of finite smodels programs such that $\text{SE}(P) \subseteq \text{SE}(Q)$.
**Complexity Analysis of WIMPL**

**Theorem.** The complement of WIMPL is in NP and NP-hard-complete, i.e., WIMPL is coNP-complete.

**Proof.** 1. It is possible to construct an NTM which
(i) chooses a model candidate \( M \subseteq H_b(P) \) for \( P \) in \( \langle P, Q \rangle \),
(ii) computes \( LM(P^M) \) in time polynomial with respect to \(|P|\),
(iii) rejects \( \langle P, Q \rangle \) if \( M \not\equiv LM(P^M) \),
(iv) computes \( LM(Q^M) \) in time polynomial with respect to \(|Q|\), and
(v) rejects \( \langle P, Q \rangle \) if \( M \equiv LM(P^M) \) and accepts it otherwise.
2. For a finite normal program \( P \),
\[ P \in \text{STABLE} \iff R(P) = \langle P, \{ a \leftarrow \neg a. \} \rangle \not\in \text{WIMPL}. \]

**Membership of SIMPL**

**Theorem.** The complement of SIMPL is in NP and NP-hard-complete, i.e., SIMPL is coNP-complete.

**Proof.** It is possible to construct an NTM which
(i) chooses an SE-interpretation \( \langle N, M \rangle \) for \( P \) in the input \( \langle P, Q \rangle \),
(ii) rejects \( \langle P, Q \rangle \) if \( M \not\equiv P \) or \( N \not\equiv P^M \),
(iii) accepts \( \langle P, Q \rangle \) if \( M \not\equiv Q \) or \( N \not\equiv Q^M \), and rejects it otherwise.
- The checks \( M \not\equiv P, N \not\equiv P^M, M \not\equiv Q \), and \( N \not\equiv Q^M \) are feasible in time polynomial with respect to \(|P| + |Q|\).
- The NTM described above has an accepting computation on \( \langle P, Q \rangle \iff \exists \langle N, M \rangle \in \text{SE}(P) \) such that \( \langle N, M \rangle \not\in \text{SE}(Q) \).

**Hardness of SIMPL**

**Theorem.** The complement of SIMPL is NP-hard-complete, i.e., SIMPL is coNP-hard-complete.

**Proof.** Consider a set of clauses \( S \) and a query atom \( c \in H_b(S) \).
1. An atom \( a \in H_b(S) \) is translated into \( R_1(a) \) using \( f \notin H_b(S) \):
\[ a \leftarrow \neg a, \neg f. \quad \neg a \leftarrow \neg a, \neg f. \quad f \leftarrow a, \neg a, \neg f. \]
2. For a clause \( l_1 \lor \ldots \lor l_n \in S, R_2(l_1 \lor \ldots \lor l_2) \) is the positive rule
\[ h^+(l_1) \leftarrow h^-(l_1), \ldots, h^-(l_n). \]
where \( h^+(a) = a, h^-(\neg a) = \neg a, h^-(\neg a) = a, \) and \( h^-(\neg a) = a. \)
Let us define \( R(S, c) = \langle R_1(H_b(S)) \cup R_2(S), R_1(H_b(S)) \cup R_2(S) \cup R_2(c) \rangle. \)
Then, for a finite set of clauses \( S \) and a query atom \( c \in H_b(S) \):
\[ S \models c \iff R(S, c) \in \text{SIMPL}. \]

**Deciding Equivalence**

**Definition.**
1. The language WEQ is the set of pairs \( \langle P, Q \rangle \) of finite smodels programs such that \( SM(P) = SM(Q) \).
2. The language SEQ is the set of pairs \( \langle P, Q \rangle \) of finite smodels programs such that \( SE(P) = SE(Q) \).

**Theorem.** Both WEQ and SEQ are coNP-complete.

**Proof.** 1. WEQ is the intersection of two coNP-complete languages, WIMPL and \( \{ \langle Q, P \rangle \mid \langle P, Q \rangle \in \text{WIMPL} \} \).
2. The reduction \( R(P) = \langle P, \{ a \leftarrow \neg a. \} \rangle \) presented above applies:
\( P \in \text{STABLE} \iff R(P) \not\in \text{WEQ} \).
The case of SEQ is proved analogously.
3. TRANSLATION-BASED VERIFICATION

- The idea is to combine two models programs \( P \) and \( Q \) into a single program \( \text{EQT}(P, Q) \) having a stable model if and only if 
  \( \exists M \in \text{SM}(P) \) such that \( M \notin \text{SM}(Q) \).
- The translation-based verification of \( P \equiv Q \) counts on 
  \( P \equiv Q \iff \text{EQT}(P, Q) \) and \( \text{EQT}(Q, P) \) have no stable models.
- It is assumed (without loss of generality) that \( \text{Hb}(P) = \text{Hb}(Q) \).
- A number of new atoms not appearing in \( \text{Hb}(P) \) are needed:
  1. an atom \( a^* \) for each atom \( a \in \text{Hb}(Q) \) to represent \( Q^M \) with respect to a potential counter-example \( M \), and
  2. atoms \( d \) and \( f \) for additional control.

**Observations about \( \text{EQT}(P, Q) \)**

- The translation \( \text{EQT}(P, Q) \) is designed to capture pairs \( \langle P, Q \rangle \) of models programs such that \( \langle P, Q \rangle \notin \text{WIMPL} \).
- To this end, the parts of \( \text{EQT}(P, Q) \) play the following roles:
  1. The rules of \( P \) capture a stable model \( M \in \text{SM}(P) \).
  2. The rules of \( Q^* \) express \( \text{LM}(Q^M) \) using \( \text{Hb}(Q)^* \).
  3. Rules of the forms \( d \leftarrow a, \sim a^* \) and \( d \leftarrow a^*, \sim a \) check whether \( M \) and \( \text{LM}(Q^M) \) differ with respect to some \( a \in \text{Hb}(Q) \).
  4. The rule \( f \leftarrow \sim d, \sim f \) excludes cases where there is no difference, i.e., \( M \neq \text{LM}(Q^M) \) is enforced.

**Theorem.** For any models programs \( P \) and \( Q \), \( \text{EQT}(P, Q) \) has a stable model \( \iff \exists M \in \text{SM}(P) \) such that \( M \notin \text{SM}(Q) \).

---

Translation for Equivalence Checking

**Definition.** The translation \( \text{EQT}(P, Q) = P \cup Q^* \cup \{d \leftarrow a, \sim a^*, f \leftarrow \sim d, \sim f \} \) where \( Q^* \) contains

1. \( a^* \leftarrow b_1^*, \ldots, b_n^*, \sim c_1, \ldots, \sim c_m \) for each basic rule (1.) in \( Q \),
2. \( a^* \leftarrow \ell \{b_1^*, \ldots, b_n^*, \sim c_1, \ldots, \sim c_m \} \) for each cardinality rule (2.) in \( Q \),
3. \( a_i^* \leftarrow b_i^*, \ldots, b_n^*, a_i, \sim c_1, \ldots, \sim c_m \) for each choice rule (3.) in \( Q \) and head atom \( a_i \in \{a_1, \ldots, a_h\} \), and
4. \( a^* \leftarrow \ell \{b_1^* = w_1, \ldots, b_n^* = w_n, \sim c_1 = v_{n+1}, \ldots, \sim c_m = v_m\} \) for each weight rule (4.) in \( Q \).

**Example**

- Let us check whether the following programs are equivalent:
  \[
  P: \{a, b\}, \quad Q: \begin{cases} a \leftarrow \sim b, \quad a \leftarrow \sim a, \sim b, \quad b \leftarrow \sim a. \\
  \end{cases}
  \]
  - The translation \( \text{EQT}(P, Q) \) consists of
    \[
    \{a, b\}, \quad a \leftarrow \sim a, \sim b, \quad a^* \leftarrow b, \quad b^* \leftarrow a.
    \]
    \[
    d \leftarrow a^*, \sim a, \quad d \leftarrow b^*, \sim b, \quad d \leftarrow a, \sim a^*, \quad d \leftarrow b, \sim b^*.
    \]
    \[
    f \leftarrow \sim d, \sim f.
    \]
  - There is \( N = \{a, b, d\} \in \text{SM}(\text{EQT}(P, Q)) \) giving rise to a counter model \( M = N \cap \text{Hb}(P) \in \text{SM}(P) \) so that \( P \neq Q \).
  - The reduct \( \text{EQT}(P, Q)^N = \{a, b, d \leftarrow a, d \leftarrow b\} \).

© 2007 TKK / TCS
Using the Translation

Corollary. For any smodels programs $P$ and $Q$,

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

Some observations and remarks follow:

- Thus, in case of a positive outcome, the verification of $P \equiv Q$ involves a two-way failing search for counter-examples.
- smodels programs that contain minimization statements are not directly covered by the translation-based method.
- But if $P$ and $Q$ are free of optimization statements and $P \equiv Q$, then they remain equivalent if extended by the same statements.

How to Use lpeq

- The weak equivalence of two smodels programs, first produced with lparse, is checked by issuing the following commands:
  
  ```
  \$ lparse p1.lp > p1.sm
  \$ lparse p2.lp > p2.sm
  \$ lpeq p1.sm p2.sm | smodels 1
  \$ lpeq p2.sm p1.sm | smodels 1
  ```

- It is also possible to verify classical equivalence (option flag -c) and strong equivalence (flag -s) and in this order.

- Programs for tests involving classical and strong equivalence must be produced with lparse's command line option `-dall`.

5. TOOL FOR EQUIVALENCE TESTING

- There is a translator called lpeq which implements the translation-based verification method described above.

- lpeq has been designed to produce EQT($P, Q$) for programs created by lparse. This may fail if too many atoms are hidden.

- The existence of potential counter-examples for $P \equiv Q$ can be checked using the smodels solver for the search.

- No special-purpose search engines need to be developed.

- The Linux binaries of lpeq and dlpeq are available at http://www.tcs.hut.fi/Software/lpeq

6. EXPERIMENTAL RESULTS

- The verification method based on the translation EQT($P, Q$) has been compared with a cross-checking approach.

- In this naive approach, the inclusion SM($P$) $\subseteq$ SM($Q$) is verified using the following algorithm:

  ```
  function Naive($P, Q$): boolean;
  var $M$: atom set;
  for $M$ in SM($P$)
  if $M \neq \text{LM}(Q^M)$ then return \perp;
  return $T$;
  ```

- The smodels solver is used to enumerate stable models whereas the stability check is done using a particular tool (testsm).

- A two-way search of counter-examples was performed in any case.
**Equivalent Programs for the $n$-Queens Problem**

- The first formulation $Q_n$ is due to Niemelä [1 999].
- The second formulation $Q'_n$ is a variant of $Q_n$ that uses choice rules and cardinality rules in addition to basic rules.

| $n$ | stable models | lpeq | naive | choices lpeq | choices naive | $|\text{Q}_n|/|\text{Q}_n'|$ |
|-----|---------------|------|-------|-------------|----------------|------------------|
| 1   | 1             | 0.000| 0.080 | 0           | 0              | 7                |
| 2   | 0             | 0.000| 0.051 | 0           | 0              | 28               |
| 3   | 2             | 0.003| 0.061 | 0           | 0              | 124              |
| 4   | 2             | 0.019| 0.120 | 0           | 2              | 300              |
| 5   | 10            | 0.042| 0.454 | 5           | 18             | 600              |
| 6   | 4             | 0.138| 0.259 | 16          | 18             | 1038             |
| 7   | 40            | 0.316| 2.340 | 40          | 84             | 1708             |
| 8   | 92            | 2.807| 6.721 | 163         | 253            | 2584             |
| 9   | 382           | 17.316| 32.032| 615         | 955            | 3720             |
| 10  | 724           | 90.866| 90.864| 2013        | 3127           | 5150             |
| 11  | 2690          | 617.279| 451.410| 11939      | 13062          | 6906             |

**Random 3-SAT Instances**

- In this experiment, random 3-SAT instances $S$ are created with a fixed clauses-to-variables ratio $\frac{c}{v} = 4$ (phase transition at 4.3).
- Instances are encoded as logic programs $P$ in terms of basic rules.
- The idea is to test $P \equiv P'$ where $P'$ is a variant of $P$ obtained by dropping one random rule from $P$.

**Observations**

- In many cases, the number of choice points and the time needed for computations is less than in the naive cross-checking approach.
- If programs being compared are likely to have no/few stable models, then the naive approach becomes superior.
- The use of hidden atoms tends to increase the complexity of equivalence checking.

**Example.** Consider the following smodels programs:

$P$: $a \leftarrow \neg b$.  $b \leftarrow \neg a$.  $c \leftarrow \neg d$.  $d \leftarrow \neg c$.

$Q$: $\{a, c\}$.

It is clear that $P \not\equiv Q$ but this is not the case if $b$ and $d$ are hidden.

**Objectives**

- You are familiar with two fundamental notions of equivalence that have been proposed for classes of programs used in ASP.
- You know the basic complexity results about verifying weak/strong equivalence in the case of normal/smodels programs.
- You understand the architecture of translation-based equivalence checking and its potential over naive cross-checking of answer sets.
- You have tried to use lpeq in practice to see whether two programs are equivalent—or differ in an intended way.
TIME TO PONDER

In this lecture, we have assumed that basic rules have a head, i.e., each constraint $\leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, c_m$ must be expressed indirectly using a new atom, say $f$, and a basic rule of the form

$$f \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_m, \neg f.$$

Consider an extension of smodels programs with constraints of the form described above (without $f$).

- Describe changes to the definition of stable models in order to cover constraints.
- How about the translation-based verification method, i.e., in which way constraints can be incorporated into $\text{EQT}(P, Q)$?