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 $\mathrm{SM}(P) = \mathrm{SM}(Q)$.
- 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* $a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m$,
  2. *cardinality rules* $a \leftarrow l\,\{b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m\}$ with $l \geq 0$,
  3. *choice rules* $\{a_1, \ldots, a_h\} \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m$, and
  4. *weight rules*
     $$a \leftarrow l\,[b_1 = w_1, \ldots, b_n = w_n, \sim c_1 = v_1, \ldots, \sim c_m = v_m]$$
     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 $M \subseteq \mathrm{Hb}(P)$, 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 $M \models \{\sim c_1, \ldots, \sim c_m\}$, **or** there is a choice rule (3.) in P such that $a \in \{a_1, \ldots, a_h\}$, $M \models a$, and $M \models \{\sim c_1, \ldots, \sim c_m\}$.
- a *cardinality rule* $a \leftarrow l'\,\{b_1, \ldots, b_n\}$ $\iff$ there is a cardinality rule (2.) in P and $l' = \max(0, l - |\{\sim c_i \mid M \models \sim c_i\}|)$,
- 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 $l' = \max(0, l - \sum_{M \models \sim c_j} v_j)$.

Definition. An interpretation $M \subseteq \mathrm{Hb}(P)$ is a *stable model* of P $\iff$ $M = \mathrm{LM}(P^M)$, 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 $\mathrm{SM}(P) = \mathrm{SM}(Q)$.

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., $\mathrm{SM}(P \cup R) = \mathrm{SM}(Q \cup R)$.

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

Examples

Consider the weak/strong equivalence of following pairs of programs:

P	Q	$P \equiv Q$?	$P \equiv_s Q$?
$a \leftarrow a.$		yes	yes
$a \leftarrow \sim b.$	$a.$	yes	no
$a \leftarrow \sim b.\ \ b \leftarrow \sim a.$	$\{a, b\}.$	no	no
$a \leftarrow b, \sim b.$		yes	yes
$a \leftarrow b.\ \ a \leftarrow \sim b.$	$a.$	yes	no
$a \leftarrow \sim a.$	$a \leftarrow b.\ \ b \leftarrow \sim a.$	yes	no

Provide a witnessing context R for the cases in which $P \not\equiv_s Q$ holds!

Characterization of Strong Equivalence

- Given an `smodels` program P, an *SE-interpretation* is a pair $\langle N, M \rangle$ of ordinary interpretations such that $N \subseteq M \subseteq \mathrm{Hb}(P)$.
- 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 $\mathrm{SE}(P) = \mathrm{SE}(Q)$, i.e., P and Q have the same SE-models.

Example. Consider $P = \{a \leftarrow b.\ \ a \leftarrow \sim b.\ \}$ and $Q = \{a.\ \}$ from the previous slide. The fact that $P \not\equiv_s Q$ is witnessed by

1. the context $R = \{b \leftarrow a.\ \}$, and
2. 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 $\mathrm{SM}(P) \subseteq \mathrm{SM}(Q)$.
2. The language SIMPL is the set of pairs $\langle P, Q \rangle$ of finite `smodels` programs such that $\mathrm{SE}(P) \subseteq \mathrm{SE}(Q)$.

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 $M \subseteq \mathrm{Hb}(P)$ for P in $\langle P, Q\rangle$,

(ii) computes $\mathrm{LM}(P^M)$ in time polynomial with respect to $||P||$,

(iii) *rejects* $\langle P, Q\rangle$ if $M \neq \mathrm{LM}(P^M)$,

(iv) computes $\mathrm{LM}(Q^M)$ in time polynomial with respect to $||Q||$, and

(v) *rejects* $\langle P, Q\rangle$ if $M = \mathrm{LM}(P^M)$ and *accepts* it otherwise.

2. For a finite *normal* program P,

$$P \in \mathsf{STABLE} \iff R(P) = \langle P, \{a \leftarrow {\sim}a.\ \}\rangle \notin \mathsf{WIMPL}. \qquad \square$$

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 $M \not\models P$ or $N \not\models P^M$,

(iii) *accepts* $\langle P, Q\rangle$ if $M \not\models Q$, or $N \not\models Q^M$, and *rejects* it otherwise.

➤ The checks $M \not\models P$, $N \not\models P^M$, $M \not\models Q$, and $N \not\models Q^M$ are feasible in time polynomial with respect to $||P|| + ||Q||$.

➤ The NTM described above has an accepting computation on $\langle P, Q\rangle \iff \exists\langle N, M\rangle \in \mathrm{SE}(P)$ such that $\langle N, M\rangle \notin \mathrm{SE}(Q)$. $\square$

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 $c \in \mathrm{Hb}(S)$.

1. An atom $a \in \mathrm{Hb}(S)$ is translated into $R_1(a)$ using $f \notin \mathrm{Hb}(S)$:

$$a \leftarrow {\sim}\overline{a}, {\sim}f. \quad \overline{a} \leftarrow {\sim}a, {\sim}f. \quad f \leftarrow a, \overline{a}, {\sim}f.$$

2. For a clause $l_1 \vee \ldots \vee l_n \in S$, $R_2(l_1 \vee \ldots \vee l_2)$ is the positive rule

$$h^+(l_1) \leftarrow h^-(l_2), \ldots, h^-(l_n).$$

where $h^+(a) = a$, $h^+(\neg a) = \overline{a}$, $h^-(a) = \overline{a}$, and $h^-(\neg a) = a$.

Let us define $R(S, c) = \langle R_1(\mathrm{Hb}(S)) \cup R_2(S), R_1(\mathrm{Hb}(S)) \cup R_2(S) \cup R_2(c)\rangle$.

Then, for a *finite* set of clauses S and a query atom $c \in \mathrm{Hb}(S)$: $S \models c \iff R(S, c) \in \mathsf{SIMPL}$. $\square$

Deciding Equivalence

Definition.

1. The language WEQ is the set of pairs $\langle P, Q\rangle$ of finite `smodels` programs such that $\mathrm{SM}(P) = \mathrm{SM}(Q)$.

2. The language SEQ is the set of pairs $\langle P, Q\rangle$ of finite `smodels` programs such that $\mathrm{SE}(P) = \mathrm{SE}(Q)$.

Theorem. Both WEQ and SEQ are coNP-complete.

Proof. 1. WEQ is the intersection of two coNP-complete languages, WIMPL and $\{\langle Q, P\rangle \mid \langle P, Q\rangle \in \mathsf{WIMPL}\}$.

2. The reduction $R(P) = \langle P, \{a \leftarrow {\sim}a.\ \}\rangle$ presented above applies: $P \in \mathsf{STABLE} \iff R(P) \notin \mathsf{WEQ}$.

The case of SEQ is proved analogously. $\square$

3. TRANSLATION-BASED VERIFICATION

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

Translation for Equivalence Checking

Definition. The translation $\mathrm{EQT}(P,Q) =$

$$P \cup Q^\star \cup \{d \leftarrow a, \sim a^\star.\ \ d \leftarrow a^\star, \sim a.\ \ \mid a \in \mathrm{Hb}(Q)\} \cup \{f \leftarrow \sim d, \sim f.\ \ \}$$

where $Q^\star$ contains

1. $a^\star \leftarrow b_1^\star, \ldots, b_n^\star, \sim c_1, \ldots, \sim c_m$ for each basic rule (1.) in Q,
2. $a^\star \leftarrow l\,\{b_1^\star, \ldots, b_n^\star, \sim c_1, \ldots, \sim c_m\}$ for each cardinality rule (2.) in Q,
3. $a_i^\star \leftarrow b_1^\star, \ldots, b_n^\star, a_i, \sim c_1, \ldots, \sim c_m$ for each choice rule (3.) in Q and head atom $a_i \in \{a_1, \ldots, a_h\}$, and
4. $a^\star \leftarrow l\,[b_1^\star = w_1, \ldots, b_n^\star = w_n, \sim c_1 = w_{n+1}, \ldots, \sim c_m = w_m]$ for each weight rule (4.) in Q.

Observations about $\mathrm{EQT}(P,Q)$

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

Theorem. For any `smodels` programs P and Q, $\mathrm{EQT}(P,Q)$ has a stable model $\iff \exists M \in \mathrm{SM}(P)$ such that $M \notin \mathrm{SM}(Q)$.

Example

- Let us check whether the following programs are equivalent:
$$P:\quad \{a,b\}.\qquad\qquad Q:\quad a \leftarrow \sim b.$$
$$a \leftarrow \sim a, \sim b.\qquad\qquad b \leftarrow \sim a.$$
- The translation $\mathrm{EQT}(P,Q)$ consists of
$$\begin{array}{llll} \{a,b\}. & a \leftarrow \sim a, \sim b. & a^\star \leftarrow \sim b. & b^\star \leftarrow \sim a. \\ d \leftarrow a^\star, \sim a. & d \leftarrow b^\star, \sim b. & d \leftarrow a, \sim a^\star. & d \leftarrow b, \sim b^\star. \\ f \leftarrow \sim d, \sim f. & & & \end{array}$$
- There is $N = \{a,b,d\} \in \mathrm{SM}(\mathrm{EQT}(P,Q))$ giving rise to a counter model $M = N \cap \mathrm{Hb}(P) \in \mathrm{SM}(P)$ so that $P \not\equiv Q$.
- The reduct $\mathrm{EQT}(P,Q)^N = \{a.\ \ b.\ \ d \leftarrow a.\ \ d \leftarrow b.\ \ \}$.

Using the Translation

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

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

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.

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 $\mathrm{EQT}(P,Q)$ 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.

  $\Longrightarrow$ 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`

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`.

6. EXPERIMENTAL RESULTS

- The verification method based on the translation $\mathrm{EQT}(P,Q)$ has been compared with a cross-checking approach.
- In this `naive` approach, the inclusion $\mathrm{SM}(P) \subseteq \mathrm{SM}(Q)$ is verified using the following algorithm:

  function Naive(P,Q): boolean;
  var M: atom set;
  for M **in** $\mathrm{SM}(P)$
    if $M \neq \mathrm{LM}(Q^M)$ **then return** $\bot$;
  return $\top$;
- 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.

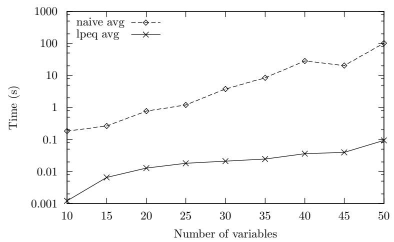
Equivalent Programs for the n-Queens Problem

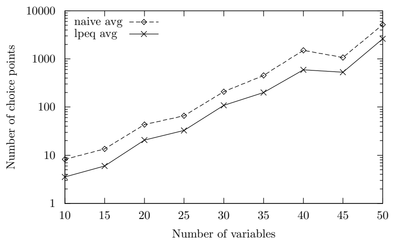
- The first formulation Q_n is due to Niemelä [1999].
- The second formulation Q'_n is a variant of Q_n that uses choice rules and cardinality rules in addition to basic rules.

n	stable models	t_{avg} (s) lpeq	t_{avg} (s) naive	choices lpeq	choices naive	$\lvert Q_n\rvert + \lvert Q'_n\rvert$	$\lvert \mathrm{EQT}(Q_n,Q'_n)\rvert + \lvert \mathrm{EQT}(Q_n,Q'_n)\rvert$
1	1	0.000	0.080	0	0	7	28
2	0	0.000	0.051	0	0	28	130
3	0	0.003	0.051	0	0	124	384
4	2	0.019	0.120	0	2	300	884
5	10	0.042	0.454	5	18	600	1718
6	4	0.136	0.259	16	18	1058	2974
7	40	0.516	2.340	40	84	1708	4740
8	92	2.967	6.721	163	253	2584	7104
9	352	17.316	32.032	615	955	3720	10154
10	724	99.866	90.694	2613	3127	5150	13978
11	2680	617.579	451.410	11939	13662	6908	18664

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:

$$P: \quad a \leftarrow \sim b. \qquad b \leftarrow \sim a. \qquad c \leftarrow \sim d. \qquad d \leftarrow \sim c.$$
$$Q: \quad \{a, c\}.$$

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* $\leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, c_m$ must be expressed indirectly using a new atom, say f, and a basic rule of the form

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

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 $\mathrm{EQT}(P, Q)$?