Answer Set Programs with Queries over Subprograms

Christoph Redl

redl@kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

July 4, 2017

Outline

Motivation

Answer Set Programming is a well-known declarative problem solving approach.

Motivation

Answer Set Programming is a well-known declarative problem solving approach.

Answer Set Programming [Gelfond and Lifschitz, 1991]

An ASP program consists of rules of form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n,$$

An interpretation I is a set of ground atoms;
it is an answer set of a ground program P, if I is a $\subseteq$-minimal model of the reduct $P^I = \{H(r) \leftarrow B^+(r) \mid r \in \Pi, I \not\models b \text{ for all } b \in B^-(r)\}$.

Semantics of non-ground programs is defined via a grounding,
i.e., replacement of all variables by all constants in all possible ways.

Motivation

Answer Set Programming is a well-known declarative problem solving approach.

Answer Set Programming [Gelfond and Lifschitz, 1991]

An ASP program consists of rules of form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n,$$

An interpretation I is a set of ground atoms;
it is an answer set of a ground program P, if I is a $\subseteq$-minimal model of the reduct $P^I = \{H(r) \leftarrow B^+(r) \mid r \in \Pi, I \not\models b \text{ for all } b \in B^-(r)\}$.

Semantics of non-ground programs is defined via a grounding,
i.e., replacement of all variables by all constants in all possible ways.

Two (Related) Restrictions

- Meta-reasoning about the answer sets of a *(sub)program* within another *(meta-)program* not inherently supported.

Motivation

Answer Set Programming is a well-known declarative problem solving approach.

Answer Set Programming [Gelfond and Lifschitz, 1991]

An ASP program consists of rules of form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n,$$

An interpretation I is a set of ground atoms;
it is an answer set of a ground program P, if I is a $\subseteq$-minimal model of the reduct $P^I = \{H(r) \leftarrow B^+(r) \mid r \in \Pi, I \not\models b \text{ for all } b \in B^-(r)\}$.

Semantics of non-ground programs is defined via a grounding,
i.e., replacement of all variables by all constants in all possible ways.

Two (Related) Restrictions

- Meta-reasoning about the answer sets of a *(sub)program* within another *(meta-)program* not inherently supported.
- Despite Σ_2^P-completeness of disjunctive ASP, solving problems from the first level of the polynomial hierarchie is sometimes tricky.

Motivation

Two (Related) Restrictions

- Meta-reasoning about the answer sets of a *(sub)program* within another *(meta-)program* not inherently supported.
- Despite Σ_2^P-completeness of disjunctive ASP, solving problems from the first level of the polynomial hierarchie within a program is difficult.

Motivation

Two (Related) Restrictions

- Meta-reasoning about the answer sets of a *(sub)program* within another *(meta-)program* not inherently supported.
- Despite Σ_2^P-completeness of disjunctive ASP, solving problems from the first level of the polynomial hierarchie within a program is difficult.

Contribution

- An encoding to decide inconsistency of a normal program within a (disjunctive) program.

Motivation

Two (Related) Restrictions

- Meta-reasoning about the answer sets of a *(sub)program* within another *(meta-)program* not inherently supported.
- Despite Σ_2^P-completeness of disjunctive ASP, solving problems from the first level of the polynomial hierarchie within a program is difficult.

Contribution

- An encoding to decide inconsistency of a normal program within a (disjunctive) program.
- An encoding for query answering over a normal program within another program.

Motivation

Two (Related) Restrictions

- Meta-reasoning about the answer sets of a *(sub)program* within another *(meta-)program* not inherently supported.
- Despite Σ_2^P-completeness of disjunctive ASP, solving problems from the first level of the polynomial hierarchie within a program is difficult.

Contribution

- An encoding to decide inconsistency of a normal program within a (disjunctive) program.
- An encoding for query answering over a normal program within another program.
- A language extension of ASP program with query atoms to be used as a new modeling technique.

Outline

The Saturation Technique

Basic idea

- Exploits disjunctions with head-cycles to solve coNP-hard problems within ASP.

The Saturation Technique

Basic idea

- Exploits disjunctions with head-cycles
  to solve coNP-hard problems within ASP.
  (Based on the hardness proof of disjunctive ASP [Eiter and Gottlob, 1995].)

The Saturation Technique and its Restrictions

The Saturation Technique

Basic idea

- Exploits disjunctions with head-cycles
  to solve coNP-hard problems within ASP.
  (Based on the hardness proof of disjunctive ASP [Eiter and Gottlob, 1995].)
- Typical use case:
  Check if a certain property holds for all objects in a certain domain.

The Saturation Technique and its Restrictions

The Saturation Technique

Basic idea

- Exploits disjunctions with head-cycles
  to solve coNP-hard problems within ASP.
  (Based on the hardness proof of disjunctive ASP [Eiter and Gottlob, 1995].)
- Typical use case:
  Check if a certain property holds for all objects in a certain domain.

Example

Check if a graph is not 3-colorable.

The Saturation Technique and its Restrictions

The Saturation Technique

Basic idea

- Exploits disjunctions with head-cycles
  to solve coNP-hard problems within ASP.
  (Based on the hardness proof of disjunctive ASP [Eiter and Gottlob, 1995].)
- Typical use case:
  Check if a certain property holds for all objects in a certain domain.

Example

Check if a graph is not 3-colorable.

Consider $P_{non3col} = F \cup P_{guess} \cup P_{check} \cup P_{sat}$ where

$$
\begin{aligned}
P_{guess} &= \{r(X) \vee g(X) \vee b(X) \leftarrow node(X)\} \\
P_{check} &= \{sat \leftarrow c(X), c(Y), edge(X,Y) \mid c \in \{r,g,b\}\} \\
P_{sat} &= \{c(X) \leftarrow node(X), sat \mid c \in \{r,g,b\}\}.
\end{aligned}
$$

The Saturation Technique and its Restrictions

The Saturation Technique

Basic idea

- Exploits disjunctions with head-cycles
  to solve coNP-hard problems within ASP.
  (Based on the hardness proof of disjunctive ASP [Eiter and Gottlob, 1995].)
- Typical use case:
  Check if a certain property holds for all objects in a certain domain.

Example

Check if a graph is not 3-colorable.

Consider $P_{non3col} = F \cup P_{guess} \cup P_{check} \cup P_{sat}$ where

$$
\begin{aligned}
P_{guess} &= \{r(X) \vee g(X) \vee b(X) \leftarrow node(X)\} \\
P_{check} &= \{sat \leftarrow c(X), c(Y), edge(X,Y) \mid c \in \{r,g,b\}\} \\
P_{sat} &= \{c(X) \leftarrow node(X), sat \mid c \in \{r,g,b\}\}.
\end{aligned}
$$

Is has the answer set $I_{sat} = A(P_{non3col})$ iff the graph F is not 3-colorable.
Otherwise its answer sets are proper subsets of I_{sat} and represent 3-colorings.

The Saturation Technique

Restrictions

- Although any problem in coNP can be polynomially reduced to brave reasoning over disjunctive ASP, the reduction is not always obvious.

The Saturation Technique

Restrictions

- Although any problem in coNP can be polynomially reduced to brave reasoning over disjunctive ASP, the reduction is not always obvious.
- In particular, the saturation encoding cannot use default-negation.
  ⇒ Checks which involve default-negations must be rewritten.

The Saturation Technique and its Restrictions

The Saturation Technique

Restrictions

- Although any problem in coNP can be polynomially reduced to brave reasoning over disjunctive ASP, the reduction is not always obvious.
- In particular, the saturation encoding cannot use default-negation.
  ⇒ Checks which involve default-negations must be rewritten.

Example

Check if a graph has no vertex cover S with size $|S| \leq k$ for some integer k.

Redl C. (TU Vienna) HEX-Programs July 4, 2017 7 / 26

The Saturation Technique and its Restrictions

The Saturation Technique

Restrictions

- Although any problem in coNP can be polynomially reduced to brave reasoning over disjunctive ASP, the reduction is not always obvious.
- In particular, the saturation encoding cannot use default-negation.
  ⇒ Checks which involve default-negations must be rewritten.

Example

Check if a graph has no vertex cover S with size $|S| \leq k$ for some integer k.

Consider P_{vc} consisting of facts F over $node$ and $edge$ and the following parts:

$$P_{guess} = \{in(X) \vee out(X) \leftarrow node(X)\}$$
$$P_{check} = \{sat \leftarrow edge(X,Y), \text{not } in(X), \text{not } in(Y);\ sat \leftarrow in(X_1), \ldots, in(X_{k+1}), X_1 \neq X_2, \ldots, X_k \neq X_{k+1}\}$$
$$P_{sat} = \{in(X) \leftarrow node(X), sat;\ out(X) \leftarrow node(X), sat\}$$

Redl C. (TU Vienna) HEX-Programs July 4, 2017 7 / 26

The Saturation Technique and its Restrictions

The Saturation Technique

Restrictions

- Although any problem in coNP can be polynomially reduced to brave reasoning over disjunctive ASP, the reduction is not always obvious.
- In particular, the saturation encoding cannot use default-negation.
  ⇒ Checks which involve default-negations must be rewritten.

Example

Check if a graph has no vertex cover S with size $|S| \leq k$ for some integer k.

Consider P_{vc} consisting of facts F over $node$ and $edge$ and the following parts:

$$P_{guess} = \{in(X) \vee out(X) \leftarrow node(X)\}$$
$$P_{check} = \{sat \leftarrow edge(X,Y), \text{not } in(X), \text{not } in(Y);\ sat \leftarrow in(X_1), \ldots, in(X_{k+1}), X_1 \neq X_2, \ldots, X_k \neq X_{k+1}\}$$
$$P_{sat} = \{in(X) \leftarrow node(X), sat;\ out(X) \leftarrow node(X), sat\}$$

This encoding does not work as desired because model $I_{sat} = A(P_{vc})$ is unstable.

Redl C. (TU Vienna) HEX-Programs July 4, 2017 7 / 26

The Saturation Technique

Example

Decide if a ground normal ASP program P is inconsistent.

The Saturation Technique and its Restrictions

The Saturation Technique

Example

Decide if a ground normal ASP program P is inconsistent.

Attempt:

$$
\begin{aligned}
P' &= \{true(a) \vee false(a) | a \in A(P)\} && (1)\\
&\cup \{inReduct(r) \leftarrow \{false(b) \mid b \in B^-(r)\} \mid r \in P\} && (2)\\
&\cup \{leastModel(a) \leftarrow inReduct(r), \{leastModel(b) \mid b \in B^+(r)\} \mid r \in P, a \in H(r)\} && (3)\\
&\cup \{noAS \leftarrow false(a), leastModel(a) \mid a \in A(P)\} && (4)\\
&\cup \{noAS \leftarrow true(a), \text{not } leastModel(a) \mid a \in A(P)\} && (5)\\
&\cup \{true(a) \leftarrow noAS;\ false(a) \leftarrow noAS \mid a \in A(P)\} && (6)\\
&\cup \{inReduct(r) \leftarrow noAS\} && (7)
\end{aligned}
$$

Redl C. (TU Vienna) HEX-Programs July 4, 2017 8 / 26

The Saturation Technique

Example

Decide if a ground normal ASP program P is inconsistent.

Attempt:

$$
\begin{aligned}
P' = & \{true(a) \vee false(a) | a \in A(P)\} && (1)\\
& \cup \{inReduct(r) \leftarrow \{false(b) \mid b \in B^-(r)\} \mid r \in P\} && (2)\\
& \cup \{leastModel(a) \leftarrow inReduct(r), \{leastModel(b) \mid b \in B^+(r)\} \mid r \in P, a \in H(r)\} && (3)\\
& \cup \{noAS \leftarrow false(a), leastModel(a) \mid a \in A(P)\} && (4)\\
& \cup \{noAS \leftarrow true(a), \mathrm{not}\, leastModel(a) \mid a \in A(P)\} && (5)\\
& \cup \{true(a) \leftarrow noAS;\ false(a) \leftarrow noAS \mid a \in A(P)\} && (6)\\
& \cup \{inReduct(r) \leftarrow noAS\} && (7)
\end{aligned}
$$

However, the comparison of the least model of the reduct to the original guess in rule uses default-negation.

The Saturation Technique and its Restrictions

The Saturation Technique

Example

Decide if a ground normal ASP program P is inconsistent.

Attempt:

$$
\begin{aligned}
P' = & \{true(a) \vee false(a) | a \in A(P)\} && (1)\\
& \cup \{inReduct(r) \leftarrow \{false(b) \mid b \in B^-(r)\} \mid r \in P\} && (2)\\
& \cup \{leastModel(a) \leftarrow inReduct(r), \{leastModel(b) \mid b \in B^+(r)\} \mid r \in P, a \in H(r)\} && (3)\\
& \cup \{noAS \leftarrow false(a), leastModel(a) \mid a \in A(P)\} && (4)\\
& \cup \{noAS \leftarrow true(a), \mathrm{not}\, leastModel(a) \mid a \in A(P)\} && (5)\\
& \cup \{true(a) \leftarrow noAS;\ false(a) \leftarrow noAS \mid a \in A(P)\} && (6)\\
& \cup \{inReduct(r) \leftarrow noAS\} && (7)
\end{aligned}
$$

However, the comparison of the least model of the reduct to the original guess in rule uses default-negation.
Eliminating negation is not straightforward.

Outline

A Meta-Program for Propositional Programs

Basic idea

- Use a meta-program M to simulate an ASP solver in disjunctive ASP.
- Encode the ASP program P to check for inconsistency as facts M^P_{gr}.
- Evaluate $M \cup M^P_{gr}$ to identify inconsistency or the answer sets of P.
- (Alternative encodings exist, see e.g. [Eiter and Polleres, 2006]).

Deciding Inconsistency of Normal Programs in Disjunctive ASP

A Meta-Program for Propositional Programs

Basic idea

- Use a meta-program M to simulate an ASP solver in disjunctive ASP.
- Encode the ASP program P to check for inconsistency as facts M^P_{gr}.
- Evaluate $M \cup M^P_{gr}$ to identify inconsistency or the answer sets of P.
- (Alternative encodings exist, see e.g. [Eiter and Polleres, 2006]).

Proposition

For any ground normal logic program P, we have that

(1) *if P is inconsistent, $M \cup M^P_{gr}$ has exactly one answer set which contains $noAS$; and*

(2) *if P is consistent, $M \cup M^P_{gr}$ has at least one answer set and none of the answer sets of M^P contains $noAS$.*

A Meta-Program for Propositional Programs

Definition

We define the *meta-program* $M = M_{extract} \cup M_{guess} \cup M_{check} \cup M_{sat}$, where:

$$
\begin{aligned}
M_{extract} = & \{atom(X) \leftarrow head(R,X);\ atom(X) \leftarrow bodyP(R,X);\ atom(X) \leftarrow bodyN(R,X)\} && (8)\\
& \cup \{rule(R) \leftarrow head(R,X);\ rule(R) \leftarrow bodyP(R,X);\ rule(R) \leftarrow bodyN(R,X)\} && (9)\\
M_{guess} = & \{true(X) \vee false(X) \leftarrow atom(X)\} && (10)\\
M_{check} = & \{inReduct(R) \leftarrow rule(R), (false(X) : bodyN(R,X))\} && (11)\\
& \cup \{outReduct(R) \leftarrow rule(R), bodyN(R,X), true(X)\} && (12)\\
& \cup \{iter(X,I) \vee niter(X,I) \leftarrow true(X), int(I);\ niter(X,I) \leftarrow false(X), int(I)\} && (13)\\
& \cup \{notApp(R) \leftarrow outReduct(R)\} && (14)\\
& \cup \{notApp(R) \leftarrow inReduct(R), bodyP(R,X), false(X)\} && (15)\\
& \cup \{notApp(R) \leftarrow head(R,X_1), bodyP(R,X_2), iter(X_1,I_1), iter(X_2,I_2), I_2 \geq I_1\} && (16)\\
& \cup \{noAS \leftarrow true(X), (notApp(R) : head(R,X))\} && (17)\\
& \cup \{noAS \leftarrow inReduct(R), head(R,X), false(X), (true(Y) : bodyP(R,Y))\} && (18)\\
& \cup \{noAS \leftarrow true(X), (niter(X,I) : int(I))\} && (19)\\
& \cup \{noAS \leftarrow iter(X,I_1), iter(X,I_2), I_1 \neq I_2\} && (20)\\
& \cup \{iter_<(X,I) \leftarrow false(X), int(I);\ iter_<(X,I_2) \leftarrow true(X), iter(X,I_1), int(I_2), I_2 > I_1\} && (21)\\
& \cup \{notApp(R) \leftarrow head(R,X_1), iter(X_1,I), I > 0, (iter_<(X_2,I) : bodyP(R,X_2))\} && (22)\\
M_{sat} = & \{true(X) \leftarrow atom(X), noAS;\ false(X) \leftarrow atom(X), noAS\} && (23)\\
& \cup \{iter(X,I) \leftarrow atom(X), int(I), noAS;\ niter(X,I) \leftarrow atom(X), int(I), noAS\} && (24)\\
& \cup \{inReduct(R) \leftarrow rule(R), noAS;\ outReduct(R) \leftarrow rule(R), noAS\} && (25)
\end{aligned}
$$

A Meta-Program for Propositional Programs

Definition

For a ground normal logic program P we let:

$$M^P_{gr} = \{int(c) \mid 0 \leq c < |A(P)|\} \cup \{head(r,h) \mid r \in P, h \in H(r)\}$$
$$\cup \{bodyP(r,b) \mid r \in P, h \in B^+(r)\} \cup \{bodyN(r,b) \mid r \in P, h \in B^-(r)\}$$

A Meta-Program for Non-Ground Programs

To lift the idea to non-ground programs, we exploit function symbols: predicates in the input program become function symbols in the meta-program.

A Meta-Program for Non-Ground Programs

To lift the idea to non-ground programs, we exploit function symbols: predicates in the input program become function symbols in the meta-program.

Definition

For a (ground or non-ground) normal logic program P we let:

$$M^P_{ng} = \{int(c) \mid 0 \leq c < |A(P)|\} \cup \{head(r(\vec{V}_r),h) \leftarrow \{head(R,d) \mid d \in B^+(r)\} \mid r \in P, h \in H(r)\}\}$$
$$\cup \{bodyP(r(\vec{V}_r),b) \leftarrow \{head(R,d) \mid d \in B^+(r)\} \mid r \in P, b \in B^+(r)\}$$
$$\cup \{bodyN(r(\vec{V}_r),b) \leftarrow \{head(R,d) \mid d \in B^+(r)\} \mid r \in P, b \in B^-(r)\}$$

A Meta-Program for Non-Ground Programs

To lift the idea to non-ground programs, we exploit function symbols: predicates in the input program become function symbols in the meta-program.

Definition

For a (ground or non-ground) normal logic program P we let:

$$M^P_{ng} = \{int(c) \mid 0 \leq c < |A(P)|\} \cup \{head(r(\vec{V}_r),h) \leftarrow \{head(R,d) \mid d \in B^+(r)\} \mid r \in P, h \in H(r)\}\}$$
$$\cup \{bodyP(r(\vec{V}_r),b) \leftarrow \{head(R,d) \mid d \in B^+(r)\} \mid r \in P, b \in B^+(r)\}$$
$$\cup \{bodyN(r(\vec{V}_r),b) \leftarrow \{head(R,d) \mid d \in B^+(r)\} \mid r \in P, b \in B^-(r)\}$$

Proposition

For any normal logic program P, we have that

(1) *if P is inconsistent, $M \cup M^P_{ng}$ has exactly one answer set which contains $noAS$; and*

(2) *if P is consistent, $M \cup M^P_{ng}$ has at least one answer set and none of the answer sets of M^P contains $noAS$.*

A Meta-Program for Non-Ground Programs

Example

Let $P = \{f : d(a);\ r_1 : q(X) \leftarrow d(X), \text{not}\, p(X);\ r_2 : p(X) \leftarrow d(X), \text{not}\, q(X)\}$.

A Meta-Program for Non-Ground Programs

Example

Let $P = \{f : d(a);\ r_1 : q(X) \leftarrow d(X), \text{not}\, p(X);\ r_2 : p(X) \leftarrow d(X), \text{not}\, q(X)\}$.

We have:

$$M^P_{ng} = \{head(f,d(a)) \leftarrow;\ head(r_1(X),q(X)) \leftarrow head(R,d(X))\}$$
$$\cup \{bodyP(r_1(X),d(X)) \leftarrow head(R,d(X));\ bodyN(r_1(X),p(X)) \leftarrow head(R,d(X))\}$$
$$\cup \{head(r_2(X),p(X)) \leftarrow head(R,d(X));\ bodyP(r_2(X),d(X)) \leftarrow head(R,d(X))\}$$
$$\cup \{bodyN(r_2(X),q(X)) \leftarrow head(R,d(X))\}$$

Outline

Query Answering over Subprograms

We reduce brave and cautious queries over subprograms
to inconsistency checking.

Query Answering over Subprograms

We reduce brave and cautious queries over subprograms
to inconsistency checking.

Observation:

Proposition

For a normal logic program P and a query q we have that

(1) $P \models_b q$ *iff* $P \cup \{\leftarrow \bar{l} \mid l \in q\}$ *is consistent; and*
(2) $P \models_c q$ *iff* $P \cup \{\leftarrow q\}$ *is inconsistent.*

Query Answering over Subprograms

We reduce brave and cautious queries over subprograms
to inconsistency checking.

Observation:

Proposition

For a normal logic program P and a query q we have that

(1) $P \models_b q$ *iff* $P \cup \{\leftarrow \bar{l} \mid l \in q\}$ *is consistent; and*
(2) $P \models_c q$ *iff* $P \cup \{\leftarrow q\}$ *is inconsistent.*

Reduction:

Proposition

For a normal logic program P and query q we have that

(1) $M \cup M_{ng}^{P \cup \{\leftarrow \bar{l} \mid l \in q\}}$ *is consistent and each answer set contains* $noAS$ *iff* $P \not\models_b q$*; and*
(2) $M \cup M_{ng}^{P \cup \{\leftarrow q\}}$ *is consistent and each answer set contains* $noAS$ *iff* $P \models_c q$.

Query Answering: Language Extension

Now we extend ASP with queries over subprograms:

Definition

A ground query atom is of form $S \vdash_t q$, where $t \in \{b, c\}$ determines the type of the query, S is a normal logic (sub)program, and q is a query over S.

Query atoms may occur in bodies of ASP programs in place of ordinary atoms.

Query Answering: Language Extension

Now we extend ASP with queries over subprograms:

Definition

A ground query atom is of form $S \vdash_t q$, where $t \in \{b, c\}$ determines the type of the query, S is a normal logic (sub)program, and q is a query over S.

Query atoms may occur in bodies of ASP programs in place of ordinary atoms.

Intuition:
$S \vdash_b q$ resp. $S \vdash_c q$ is true (wrt. all interpretations I of P) if $S \models_b q$ resp. $S \models_c q$.

Query Answering: Language Extension

Now we extend ASP with queries over subprograms:

Definition

A ground query atom is of form $S \vdash_t q$, where $t \in \{b, c\}$ determines the type of the query, S is a normal logic (sub)program, and q is a query over S.

Query atoms may occur in bodies of ASP programs in place of ordinary atoms.

Intuition:
$S \vdash_b q$ resp. $S \vdash_c q$ is true (wrt. all interpretations I of P) if $S \models_b q$ resp. $S \models_c q$.

Formally the semantics of such a program P uses the following translation:

$$[P] = P|_{S \vdash_t q \rightarrow noAS_{S \vdash_t q}} \cup \bigcup_{S \vdash_b q \text{ in } P} \left(M \cup M_{ng}^{S \cup \{\leftarrow \bar{l} \mid l \in q\}}\right)\Big|_{a \rightarrow a_{S \vdash_b q}} \cup \bigcup_{S \vdash_c q \text{ in } P} \left(M \cup M_{ng}^{S \cup \{\leftarrow q\}}\right)\Big|_{a \rightarrow a_{S \vdash_c q}}$$

Query Answering: Language Extension

Proposition

For a logic program P with query atoms, the answer sets of P and $[P]$, projected to the atoms in P, coincide.

Example

Suppose P_{guess} guesses all edge selections in a graph and P_{check} derives $invalid$ if the current selection is not a Hamiltonian cycle.

Query Answering over Subprograms

Query Answering: Language Extension

Proposition

For a logic program P with query atoms, the answer sets of P and $[P]$, projected to the atoms in P, coincide.

Example

Suppose P_{guess} guesses all edge selections in a graph and P_{check} derives $invalid$ if the current selection is not a Hamiltonian cycle.

Then

$$P = \{noHamiltonian \leftarrow P_{guess} \cup P_{check} \vdash_c invalid\}$$

(and thus $[P]$) has an answer set containing $noHamiltonian$ if and only if the graph at hand does not contain a Hamiltonian cycle.

Otherwise the program has at least one answer set but none of the answer sets contains atom $noHamiltonian$.

Query Answering: Checking Conditions with Default-Negation

In general:

Steps

1. Let program P_{guess} span a search space of all objects to check.
2. Let P_{check} check if the current guess satisfies the criteria and derive atom ok in this case.
3. Instead manually saturating whenever ok is true, just checks if ok is cautiously entailed by $P_{guess} \cup P_{check}$.

Outline

Discussion

Alternative Approaches

- Nested HEX-programs [Eiter et al., 2013]:
  Baesd on the ASP-extension of HEX-programs (with access to external sources) rather than plain ASP.

Discussion

Alternative Approaches

- Nested HEX-programs [Eiter et al., 2013]:
  Baesd on the ASP-extension of HEX-programs (with access to external sources) rather than plain ASP.
- Modular ASP [Tari et al., 2005]:
  Programs consisting of components but no compilation into a single program.

Discussion

Alternative Approaches

- Nested HEX-programs [Eiter et al., 2013]:
  Baesd on the ASP-extension of HEX-programs (with access to external sources) rather than plain ASP.
- Modular ASP [Tari et al., 2005]:
  Programs consisting of components but no compilation into a single program.
- Manifold programs [Faber and Woltran, 2011]:
  Compilation-based using weak constraints.

Discussion

Alternative Approaches

- Nested HEX-programs [Eiter et al., 2013]:
  Baesd on the ASP-extension of HEX-programs (with access to external sources) rather than plain ASP.
- Modular ASP [Tari et al., 2005]:
  Programs consisting of components but no compilation into a single program.
- Manifold programs [Faber and Woltran, 2011]:
  Compilation-based using weak constraints.
- Stable-unstable semantics [Bogaerts et al., 2016]:
  Similar goal (isolating the oracle program) and generalize to the PH,
  but no dedicated query atoms.

Discussion

Alternative Approaches

- Nested HEX-programs [Eiter et al., 2013]:
  Baesd on the ASP-extension of HEX-programs (with access to external sources) rather than plain ASP.
- Modular ASP [Tari et al., 2005]:
  Programs consisting of components but no compilation into a single program.
- Manifold programs [Faber and Woltran, 2011]:
  Compilation-based using weak constraints.
- Stable-unstable semantics [Bogaerts et al., 2016]:
  Similar goal (isolating the oracle program) and generalize to the PH,
  but no dedicated query atoms.
- Encoding is also related to debugging approaches
  (e.g. [Gebser et al., 2008, Oetsch et al., 2010]):
  Rather than explaining inconsistency we exploit it for query answering.

Outline

Conclusion

Two Restriction of Answer Set Programming

- No meta-reasoning over collections of answer sets.
- Limitations of the saturation technique:
  difficult to use for ASP laymen and restricted use of default-negation.

Conclusion

Two Restriction of Answer Set Programming

- No meta-reasoning over collections of answer sets.
- Limitations of the saturation technique:
  difficult to use for ASP laymen and restricted use of default-negation.

Contribution and Solution

- Encoding for deciding inconsistency of a normal program.
- Encoding for query answering over a normal program.
- A language extension of ASP program with dedicated query atoms.
- More user-friendly alternative to saturation.

Conclusion

Two Restriction of Answer Set Programming

- No meta-reasoning over collections of answer sets.
- Limitations of the saturation technique:
  difficult to use for ASP laymen and restricted use of default-negation.

Contribution and Solution

- Encoding for deciding inconsistency of a normal program.
- Encoding for query answering over a normal program.
- A language extension of ASP program with dedicated query atoms.
- More user-friendly alternative to saturation.

Future Work

- Extension to non-ground queries.
- Implementation and application for nested HEX-program evaluation.

References I

Bogaerts, B., Janhunen, T., and Tasharrofi, S. (2016).
Stable-unstable semantics: Beyond NP with normal logic programs.
TPLP, 16(5-6):570–586.

Eiter, T. and Gottlob, G. (1995).
On the computational cost of disjunctive logic programming: Propositional case.
Ann. Math. Artif. Intell., 15(3-4):289–323.

Eiter, T., Krennwallner, T., and Redl, C. (2013).
HEX-programs with nested program calls.
In Tompits, H., editor, *Proceedings of the Nineteenth International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2011)*, volume 7773 of *LNAI*, pages 1–10. Springer.

Eiter, T. and Polleres, A. (2006).
Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications.
TPLP, 6(1-2):23–60.

References II

Faber, W. and Woltran, S. (2011).
Manifold answer-set programs and their applications.
In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *Lecture Notes in Computer Science*, pages 44–63. Springer.

Gebser, M., Pührer, J., Schaub, T., and Tompits, H. (2008).
A meta-programming technique for debugging answer-set programs.
In *AAAI*, pages 448–453. AAAI Press.

Gelfond, M. and Lifschitz, V. (1991).
Classical Negation in Logic Programs and Disjunctive Databases.
New Generation Computing, 9(3–4):365–386.

Oetsch, J., Pührer, J., and Tompits, H. (2010).
Catching the ouroboros: On debugging non-ground answer-set programs.
TPLP, 10(4-6):513–529.

References III

Tari, L., Baral, C., and Anwar, S. (2005).
A language for modular answer set programming: Application to ACC tournament scheduling.
In Vos, M. D. and Provetti, A., editors, *Answer Set Programming, Advances in Theory and Implementation, Proceedings of the 3rd Intl. ASP'05 Workshop, Bath, UK, September 27-29, 2005*, volume 142 of *CEUR Workshop Proceedings*. CEUR-WS.org.