First-order answer set programming as constructive proof search

Paweł Urzyczyn
(joint work with Aleksy Schubert)

Logic Programming

A logic program:

$X :- \ Y, Z;$

$U :- \ Z, V;$

$Y :- \ Z;$

$Z :- \ .$

Logic Programming

A logic program:

$X :- Y, Z;$

$U :- Z, V;$

$Y :- Z;$

$Z :- \quad .$

The meaning of this program is the set of derived atoms.

Logic Programming

A logic program:

$X :- Y, Z;$

$U :- Z, V;$

$Y :- Z;$

$Z :- \quad .$

The meaning of this program is the set of derived atoms.

They are: Z,

Logic Programming

A logic program:

$X :- Y, Z;$

$U :- Z, V;$

$Y :- Z;$

$Z :- \ .$

The meaning of this program is the set of derived atoms.

They are: Z, Y,

Logic Programming

A logic program:

$X :- Y, Z;$

$U :- Z, V;$

$Y :- Z;$

$Z :- \ .$

The meaning of this program is the set of derived atoms.

They are: Z, Y, X,

Logic Programming

A logic program:

$X \mathrel{:-} Y, Z;$

$U \mathrel{:-} Z, V;$

$Y \mathrel{:-} Z;$

$Z \mathrel{:-} \ .$

The meaning of this program is the set of derived atoms.

They are: Z, Y, X,

but neither U nor V.

A logic program with negations

$$X :- Y, Z, U;$$

$$Y :- \neg Z;$$

$$Y :- Z;$$

$$Z :- \neg X, \neg U;$$

$$U :- Y, \neg Z;$$

Which atoms are “derivable” (forced true)?

A logic program with negations

$X :- Y, Z, U;$

$Y :- \neg Z;$

$Y :- Z;$

$Z :- \neg X, \neg U;$

$U :- Y, \neg Z;$

Which atoms are “derivable” (forced true)?

There are many possible answers. One is...

Answer Set Programming

Kolaitis, Papadimitriou,
Why not Negation by Fixpoint?, PODS'88:

1. Guess which atoms should be forced.
2. Verify that exactly these atoms are forced.

Bad guess

$$X :- Y, Z, U;$$

$$Y :- \neg Z;$$

$$Y :- Z;$$

$$Z :- \neg X, \neg U;$$

$$U :- Y, \neg Z;$$

Guess 1: Atoms X and Z are forced, atoms Y and U are not.

Bad guess

$X :- Y, Z, U;$

$Y :- \neg Z;$ Red clauses are invalid.

$Y :- Z;$

$Z :- \neg X, \neg U;$

$U :- Y, \neg Z.$

Guess 1: Atoms X and Z are forced, atoms Y and U are not.

Bad guess

$X :- Y, Z, U;$

$Y :- \neg Z;$ Red clauses are invalid.

$Y :- Z;$ Nothing can be derived: this guess is wrong.

$Z :- \neg X, \neg U;$

$U :- Y, \neg Z.$

Guess 1: Atoms X and Z are forced, atoms Y and U are not.

Good guess

$X :- Y, Z, U;$

$Y :- \neg Z;$

$Y :- Z;$

$Z :- \neg X, \neg U;$

$U :- Y, \neg Z;$

Guess 2: Atoms Y and U are forced, atoms X and Z are not.

Good guess

$X :- Y, Z, U;$ Green assumptions are satisfied.

$Y :- \neg Z;$

$Y :- Z;$

$Z :- \neg X, \neg U;$ This clause is invalid.

$U :- Y, \neg Z.$

Guess 2: Atoms Y and U are forced, atoms X and Z are not.

Good guess

$X :- Y, Z, U;$ Green assumptions are satisfied.

$Y :- \neg Z;$ Y can be derived

$Y :- Z;$

$Z :- \neg X, \neg U;$ This clause is invalid.

$U :- Y, \neg Z.$

Guess 2: Atoms Y and U are forced, atoms X and Z are not.

Good guess

$X :- Y, Z, U;$	Green assumptions are satisfied.
$Y :- \neg Z;$	Y can be derived
$Y :- Z;$	
$Z :- \neg X, \neg U;$	This clause is invalid.
$U :- Y, \neg Z.$	U can be derived

Guess 2: Atoms Y and U are forced, atoms X and Z are not.

Another good guess

$X :- Y, Z, U;$

$Y :- \neg Z;$

$Y :- Z;$

$Z :- \neg X, \neg U;$

$U :- Y, \neg Z;$

Guess 3: Atoms Z, Y are forced, others are not.

Another good guess

$X :- Y, Z, U;$ Green assumptions are satisfied.

$Y :- \neg Z;$ Red clauses are invalid.

$Y :- Z;$

$Z :- \neg X, \neg U;$

$U :- Y, \neg Z;$

Guess 3: Atom Z, Y are forced, others are not.

∘

Another good guess

$X :- Y, Z, U;$ Green assumptions are satisfied.

$Y :- \neg Z;$ Red clauses are invalid.

$Y :- Z;$

$Z :- \neg X, \neg U;$ Z can be derived

$U :- Y, \neg Z;$

Guess 3: Atom Z, Y are forced, others are not.

∘

Another good guess

$X :- Y, Z, U;$ Green assumptions are satisfied.

$Y :- \neg Z;$ Red clauses are invalid.

$Y :- Z;$ Y can be derived

$Z :- \neg X, \neg U;$ Z can be derived

$U :- Y, \neg Z;$

Guess 3: Atom Z, Y are forced, others are not.

Stable models

$X :- Y, Z, U;$

$Y :- \neg Z;$

$Y :- Z;$

$Z :- \neg X, \neg U;$

$U :- Y, \neg Z;$

This program has two *stable models* $\{U, Y\}$ and $\{Z, Y\}$.

It *entails* Y under stable model semantics.

Write $P \models_{\mathsf{sms}} Y$.

Main definition

Given a program P and model $\mathfrak{M}$, define program $P_{\mathfrak{M}}$ without negations, as follows:

- For $X \notin \mathfrak{M}$, delete $\neg X$ from the rhs of all clauses of P;
- For $X \in \mathfrak{M}$, delete all clauses of P with $\neg X$ at the rhs.

The model $\mathfrak{M}$ is *stable* (is an *answer set*) for P, when exactly the atoms in $\mathfrak{M}$ are derivable from $P_{\mathfrak{M}}$.

∘

ASP and intuitionistic logic

Characterizing ASP in terms of two-element Kripke models:

- David Pearce. Stable inference as intuitionistic validity. *The Journal of Logic Programming*, 38(1):79–91, 1999.

ASP and intuitionistic logic

Characterizing ASP in terms of two-element Kripke models:

- David Pearce. Stable inference as intuitionistic validity. *The Journal of Logic Programming*, 38(1):79–91, 1999.

Can this be done in terms of *intuitionistic logic* per se?

The plan

- Given program P and atom Ω, write a formula φ such that $P \models_{\mathsf{sms}} \Omega$ if and only if φ is provable.

The plan

- Given program P and atom Ω, write a formula φ such that $P \models_{\mathsf{sms}} \Omega$ if and only if φ is provable.
- Define a translation backward (for a class of formulas within co-NP).

The plan

- Given program P and atom Ω, write a formula φ such that $P \models_{\mathsf{sms}} \Omega$ if and only if φ is provable.
- Define a translation backward (for a class of formulas within co-NP).
- Do the same for datalog using a co-Nexptime complete class of first-order formulas.

Forward translation

- Define formula φ so that $P \models_{\mathsf{sms}} \Omega$ iff φ is provable.

Forward translation

- Define formula φ so that $P \models_{\mathsf{sms}} \Omega$ iff φ is provable.

We construct $\varphi = \psi_1 \to \psi_2 \to \cdots \to \psi_m \to 0$.

Forward translation

- Define formula φ so that $P \models_{\mathsf{sms}} \Omega$ iff φ is provable.

We construct $\varphi = \psi_1 \to \psi_2 \to \cdots \to \psi_m \to 0$.

The entailment should hold *in every model*. For example:

Assume P has only 4 atoms X, Y, Z, and Ω. Then among the axioms ψ_i there will be formulas:

$(\overline{X} \to 1) \to (X \to 1) \to 0 \qquad (\overline{Y} \to 2) \to (Y \to 2) \to 1$

$(\overline{Z} \to 3) \to (Z \to 3) \to 2 \qquad (\overline{\Omega} \to 4) \to (\Omega \to 4) \to 3$

No other axiom has $0,1,2,3$ as target.

Forward translation

- Define formula φ so that $P \models_{\mathsf{sms}} \Omega$ iff φ is provable.

We construct $\varphi = \psi_1 \to \psi_2 \to \cdots \to \psi_m \to 0$.

The entailment should hold *in every model*. For example:

Assume P has only 4 atoms X, Y, Z, and Ω. Then among the axioms ψ_i there will be formulas:

$(\overline{X} \to 1) \to (X \to 1) \to 0$ $\qquad$ $(\overline{Y} \to 2) \to (Y \to 2) \to 1$

$(\overline{Z} \to 3) \to (Z \to 3) \to 2$ $\qquad$ $(\overline{\Omega} \to 4) \to (\Omega \to 4) \to 3$

No other axiom has $0,1,2,3$ as target.

To prove 0 from the initial assumptions one must derive 4 under an arbitrary choice of overlined and non-overlined atoms.

How to ensure entailment

The entailment $P \models_{\mathsf{sms}} \Omega$ means that one of the three cases holds in every model $\mathfrak{M}$:

Ω) either $\Omega \in \mathfrak{M}$, or

A) the model is unstable because too much is derivable ($P_{\mathfrak{M}}$ is unsound for $\mathfrak{M}$), or

B) the model is unstable because too little is derivable ($P_{\mathfrak{M}}$ is incomplete for $\mathfrak{M}$).

Three more axioms: $\quad \Omega \rightarrow 4, \quad A \rightarrow 4, \quad B \rightarrow 4.$

Proving unsoundness

We include in our formula the following axioms:

- $\overline{X}_i \to X_i! \to A$, for every atom X_i of P;
- all clauses of P where X_i is renamed as $X_i!$ and $\neg X_i$ is renamed as $\overline{X}_i$.

Let $\mathfrak{M}$ be the model corresponding to the present context.

One can prove $X_i!$ if and only if the program $P_{\mathfrak{M}}$ derives X_i.

Thus A is provable if and only if $P_{\mathfrak{M}}$ derives some $X_i \notin \mathfrak{M}$.

$\circ$

Proving incompleteness

One proves B iff some $X_i \in \mathfrak{M}$ cannot be derived by $P_{\mathfrak{M}}$.

We include in our formula the axiom:

- $X_i \rightarrow X_i? \rightarrow B$,

for every atom X_i of P.

Every proof of $X_i?$ represents a *refutation* of X_i. How?

Proving non-provability

Suppose we have only two clauses with target X:

K1: $X :- Y, Z, \neg U$;   K2: $X :- V, U, \neg Z$.

Then we have the axiom:

$(X? \to K_1) \to (X? \to K_2) \to X?$

Proving non-provability

Suppose we have only two clauses with target X:

K1: $X :- Y, Z, \neg U$; K2: $X :- V, U, \neg Z$.

Then we have the axiom:

$(X? \to K_1) \to (X? \to K_2) \to X?$

In particular we need to derive K_1 (using $X?$).

Proving non-provability

Suppose we have only two clauses with target X:

K1: $X :- Y, Z, \neg U$; K2: $X :- V, U, \neg Z$.

Then we have the axiom:

$(X? \to K_1) \to (X? \to K_2) \to X?$

In particular we need to derive K_1 (using $X?$).

For that purpose we have axioms:

$Y?{\to}K_1$, $Z?{\to}K_1$, $U{\to}K_1$,

Proving non-provability

Suppose we have only two clauses with target X:

K1: $X :- Y, Z, \neg U$; K2: $X :- V, U, \neg Z$.

Then we have the axiom:

$(X? \to K_1) \to (X? \to K_2) \to X?$

In particular we need to derive K_1 (using $X?$).

For that purpose we have axioms:

$Y?{\to}K_1,\ \ Z?{\to}K_1,\ \ U{\to}K_1,$

If $U \in \mathfrak{M}$ then K_1 is provable: clause K1 cannot derive X.

Proving non-provability

Suppose we have only two clauses with target X:

K1: $X :- Y, Z, \neg U$; K2: $X :- V, U, \neg Z$.

Then we have the axiom:

$(X? \to K_1) \to (X? \to K_2) \to X?$

In particular we need to derive K_1 (using $X?$).

For that purpose we have axioms:

$Y? {\to} K_1$, $Z? {\to} K_1$, $U {\to} K_1$,

If $U \in \mathfrak{M}$ then K_1 is provable: clause K1 cannot derive X.

Otherwise we can try to prove e.g. $Y?$
(Clause K1 can't be used if Y is not derivable.)

Termination

Can this process go forever?

Termination

Can this process go forever?

To prove $X?$ we attempt to prove $Y?$,
this makes us try to prove $Z?$ etc.

Termination

Can this process go forever?

To prove $X?$ we attempt to prove $Y?$,
this makes us try to prove $Z?$ etc.

Recall the axiom: $(X? \to K_1) \to (X? \to K_2) \to X?$

Each time we address a proof goal $X?$
we add the assumption $X?$

Termination

Can this process go forever?

To prove $X?$ we attempt to prove $Y?$,
this makes us try to prove $Z?$ etc.

Recall the axiom: $(X? \to K_1) \to (X? \to K_2) \to X?$

Each time we address a proof goal $X?$
we add the assumption $X?$

Should $X?$ appear as a proof goal again, we win instantly.

Termination

Can this process go forever?

To prove $X?$ we attempt to prove $Y?$,
this makes us try to prove $Z?$ etc.

Recall the axiom: $(X? \to K_1) \to (X? \to K_2) \to X?$

Each time we address a proof goal $X?$
we add the assumption $X?$

Should $X?$ appear as a proof goal again, we win instantly.

Note that such judgments are only classically valid:

$\ldots, (X? \to K_1) \to (X? \to K_2) \to X?, \cdots \vdash X?$

$\circ$

Backward translation

ASP entailment is NP-complete. Intutionistic propositional logic is Pspace-complete. Backward translation is only possible for formulas of a simple shape (pseudo-DNF formulas).

Backward translation

ASP entailment is NP-complete. Intutionistic propositional logic is Pspace-complete. Backward translation is only possible for formulas of a simple shape (pseudo-DNF formulas).

The principle of backward translation: for a given formula φ write a program P so that

φ is not provable if and only if P has a stable model.

Slogan: stable model $\equiv$ refutation.

The first-order case

The plan

First-order datalog ASP is Nexptime-complete.
The appropriate first-order fragment should be co-Nexptime-complete.

- We translate the entailment $P \models_{\mathsf{sms}} \Omega$ into a first-order Σ_1 formula φ with nullary targets.
- Such formulas can be replaced by monadic Σ_1 formulas (with only unary predicates).
- Refutability of bounded-arity Σ_1 formulas reduces to ASP. (Refutation soup $\Rightarrow$ stable model.)

The class Σ_1

We only consider formulas written with $\forall$ and $\to$.

Positions of $\forall$ are classified as "positive" (covariant) and "negative" (contravariant). The class Σ_1 has $\forall$ only at negative positions.

The class Σ_1

We only consider formulas written with $\forall$ and $\to$.

Positions of $\forall$ are classified as "positive" (covariant) and "negative" (contravariant). The class Σ_1 has $\forall$ only at negative positions.

Provability of Σ_1 formulas is Expspace-complete in general.

The class Σ_1

We only consider formulas written with $\forall$ and $\to$.

Positions of $\forall$ are classified as "positive" (covariant) and "negative" (contravariant). The class Σ_1 has $\forall$ only at negative positions.

Provability of Σ_1 formulas is Expspace-complete in general.

Provability of Σ_1 formulas with nullary targets is co-Nexptime-complete.

The class Σ_1

We only consider formulas written with $\forall$ and $\to$.

Positions of $\forall$ are classified as "positive" (covariant) and "negative" (contravariant). The class Σ_1 has $\forall$ only at negative positions.

Provability of Σ_1 formulas is Expspace-complete in general.

Provability of Σ_1 formulas with nullary targets is co-Nexptime-complete.

Same for Σ_1 formulas with bounded-arity predicates.

The class Σ_1

We only consider formulas written with $\forall$ and $\to$.

Positions of $\forall$ are classified as "positive" (covariant) and "negative" (contravariant). The class Σ_1 has $\forall$ only at negative positions.

Provability of Σ_1 formulas is Expspace-complete in general.

Provability of Σ_1 formulas with nullary targets is co-Nexptime-complete.

Same for Σ_1 formulas with bounded-arity predicates.

In fact all we need is this pattern:

$$\vec{\forall}(\ldots) \to \vec{\forall}(\ldots) \to \cdots \to \vec{\forall}(\ldots) \to \mathbf{a}$$

Forward translation: first-order case

Given program P and atom Ω, write a formula

$\varphi = \psi_1 \to \psi_2 \to \cdots \to \psi_m \to \mathit{loop}$

such that $P \models_{\mathsf{sms}} \Omega$.

Forward translation: first-order case

Given program P and atom Ω, write a formula

$\varphi = \psi_1 \to \psi_2 \to \cdots \to \psi_m \to \mathit{loop}$

such that $P \models_{\mathsf{sms}} \Omega$.

Model construction in every branch of the proof:

$\forall \vec{z}((\mathrm{R}(\vec{z}) \to \mathit{loop}) \to (\overline{\mathrm{R}}(\vec{z}) \to \mathit{loop}) \to \mathit{loop})$

Forward translation: first-order case

Given program P and atom Ω, write a formula

$\varphi = \psi_1 \to \psi_2 \to \cdots \to \psi_m \to \mathit{loop}$

such that $P \models_{\mathsf{sms}} \Omega$.

Model construction in every branch of the proof:

$\forall \vec{z}((\mathrm{R}(\vec{z}) \to \mathit{loop}) \to (\overline{\mathrm{R}}(\vec{z}) \to \mathit{loop}) \to \mathit{loop})$

Case dispatch as before:

$\Omega \to \mathit{loop}$, $\mathrm{A} \to \mathit{loop}$, and $\mathrm{B} \to \mathit{loop}$

Proving unsoundness with nullary targets

Instead of $\overline{X}_i \to X_i! \to A$ we use axioms

$$\forall \vec{x}.\, \overline{\mathrm{R}}(\vec{x}) \to (\mathrm{R}!(\vec{x}) \to \bullet) \to \mathrm{A},$$

That is, we prove $\bullet$, accumulating knowledge of derivation goals $\mathrm{R}!(\vec{c})$ visited so far. For a clause like

$\mathrm{R}(\vec{x}) \mathrel{:-} \mathrm{P}(\vec{x}), \mathrm{Q}(\vec{x}), \neg\mathrm{S}(\vec{x})$

we have an axiom of the form:

$\forall \vec{x}.\, \mathrm{R}!(\vec{x}) \to (\mathrm{P}!(\vec{x}) \to \bullet) \to (\mathrm{Q}!(\vec{x}) \to \bullet) \to \overline{\mathrm{S}}(\vec{x}) \to \bullet,$

Proof succeeds when we arrive at a fact (no more subgoals).

Proving incompletenenss with nullary targets

The basic axiom scheme is

$\forall\vec{x}.\, \mathrm{R}(\vec{x}) \to (\mathrm{R?}(\vec{x}) \to \circ) \to \mathrm{B}$.

(Oversimplified) axiom scheme for $\mathrm{R?}(\vec{x})$ is

$\forall\vec{x}.\mathrm{R?}(\vec{x}) \to (K_1(\vec{x}) \to \overline{K}_1) \to \cdots \to (K_n(\vec{x}) \to \overline{K}_n) \to \circ$,

where K_i are clauses of target $\mathrm{R?}(\vec{x})$.
This time we accumulate a history of a refuting play.

$\circ$

Proving incompletenenss, cont'd

If clause K_i is e.g. $\mathrm{R}(\vec{x}) :- \mathrm{P}(\vec{x}), \mathrm{Q}(\vec{x}), \neg\mathrm{S}(\vec{x})$

then we have this axiom, where RP "remembers" a single refutation step.

$\forall\vec{x}.\, K_i(\vec{x}) \to (\mathrm{P?}(\vec{x}) \to \mathrm{RP}(\vec{x}) \to \circ) \to \overline{K}_i$

This "memory" is made transitive with axioms:

$\forall\vec{x}\vec{y}\vec{z}(\mathrm{RP}(\vec{x},\vec{y}) \to \mathrm{PQ}(\vec{y},\vec{z}) \to (\mathrm{RQ}(\vec{x},\vec{z}) \to \circ) \to \circ)$

The refuter can win by discovering a loop:

$\forall\vec{x}(\mathrm{PP}(\vec{x},\vec{x}) \to \circ)$.

Backward translation

Given a monadic formula φ define a program P so that stable models of P represent refutations of φ.

Backward translation

Given a monadic formula φ define a program P so that stable models of P represent refutations of φ.

Refutations must be made concise (exponential size).