Complexity of Super-Coherence Problems in ASP

Mario Alviano[1], Wolfgang Faber[1] and Stefan Woltran[2]

[1] University of Calabria, Italy
[2] Vienna University of Technology, Austria

Pescara, 1 September 2011
CILC 2011

Outline

1. Super-coherent ASP Programs
   - Introduction, Motivation and Contribution
   - Definitions and Examples
2. Main Results
   - Proof Sketch
   - Consequence of our results

Introduction

- Answer Set Programming (ASP)
  - Logic Programming under stable model semantics
  - Associates each program with a (possibly empty) set of stable models

Coherence Problem

Deciding whether a program has at least one stable model.

Super-coherence Problem

Deciding whether a program P is such that $P \cup F$ is coherent for each set F of facts.

Why Studying Super-coherence?

1. Dynamic Magic Sets only apply to super-coherent programs [A., Faber; 2010]
2. Super-coherent programs are non-constraining
   - Adding extensional information to these programs will always result in stable models
   - Important for modular evaluation: If the top-part of a split program is super-coherent, coherence of the full program can be checked by only considering the bottom-part
3. Incoherent programs are one of the main criticisms of ASP (especially in database theory)
   - Coherence has been of interest for quite some time
   - Super-coherence emerges naturally when a fixed program and a variable database are considered

Why Studying Super-coherence?

1. Dynamic Magic Sets only apply to super-coherent programs [A., Faber; 2010]
2. Super-coherent programs are non-constraining
   - Adding extensional information to these programs will always result in stable models
   - Important for modular evaluation: If the top-part of a split program is super-coherent, coherence of the full program can be checked by only considering the bottom-part
3. Incoherent programs are one of the main criticisms of ASP (especially in database theory)
   - Coherence has been of interest for quite some time
   - Super-coherence emerges naturally when a fixed program and a variable database are considered

Why Studying Super-coherence?

1. Dynamic Magic Sets only apply to super-coherent programs [A., Faber; 2010]
2. Super-coherent programs are non-constraining
    - Adding extensional information to these programs will always result in stable models
    - Important for modular evaluation: If the top-part of a split program is super-coherent, coherence of the full program can be checked by only considering the bottom-part
3. Incoherent programs are one of the main criticisms of ASP (especially in database theory)
    - Coherence has been of interest for quite some time
    - Super-coherence emerges naturally when a fixed program and a variable database are considered

Main Contribution

What is the complexity of deciding super-coherence of ASP programs?

- Recall: deciding coherence is
  - Σ_2^P-complete for disjunctive programs
  - NP-complete for non-disjunctive programs

Contributions

- We prove Π_3^P-completeness in the disjunctive case
- We prove Π_2^P-completeness in the non-disjunctive case

Note: We focus on propositional programs.

Outline

1. Super-coherent ASP Programs
   - Introduction, Motivation and Contribution
   - Definitions and Examples
2. Main Results
   - Proof Sketch
   - Consequence of our results

ASP Syntax

An ASP program P is a finite set of rules r of the form

$$p_1 \lor \cdots \lor p_n \leftarrow q_1, \ldots, q_j, \textit{not } q_{j+1}, \ldots, \textit{not } q_m.$$

- $At(P)$: the set of atoms appearing in P

Example

$$\begin{aligned}
\text{“}NP \neq P\text{”} \lor \text{“}NP = P\text{”} &\leftarrow \\
\text{“}NP = P\text{”} &\leftarrow \text{“polynomial algorithm for SAT”} \\
\text{“PH collapses”} &\leftarrow \text{“}NP = P\text{”} \\
\text{“ASP harder than SAT”} &\leftarrow \textit{not } \text{“PH collapses”}
\end{aligned}$$

ASP Syntax

An ASP program P is a finite set of rules r of the form

$$p_1 \lor \cdots \lor p_n \leftarrow q_1, \ldots, q_j, \mathit{not}\ q_{j+1}, \ldots, \mathit{not}\ q_m.$$

- $At(P)$: the set of atoms appearing in P

Example

$$\begin{array}{rcl} \text{“}NP \neq P\text{”} \lor \text{“}NP = P\text{”} & \leftarrow & \\ \text{“}NP = P\text{”} & \leftarrow & \text{“}\textit{polynomial algorithm for SAT}\text{”} \\ \text{“}\textit{PH collapses}\text{”} & \leftarrow & \text{“}NP = P\text{”} \\ \text{“}\textit{ASP harder than SAT}\text{”} & \leftarrow & \mathit{not}\ \text{“}\textit{PH collapses}\text{”} \end{array}$$

ASP Semantics

Let P be an ASP program and $I \subseteq At(P)$ an interpretation.

- Atoms in I are true; atoms not in I are false
- A rule is satisfied if at least one head atom is true whenever all body literals are true
- If all rules of P are satisfied, then I is a model of P

Definition (Stable Models)

- Compute the FLP reduct — P^I
  - Delete from P every rule with a false body literal
- I is a stable model if I is a subset-minimal model of P^I
- $\mathcal{SM}(P)$: the set of all stable models of P

ASP Semantics

Let P be an ASP program and $I \subseteq At(P)$ an interpretation.

- Atoms in I are true; atoms not in I are false
- A rule is satisfied if at least one head atom is true whenever all body literals are true
- If all rules of P are satisfied, then I is a model of P

Definition (Stable Models)

- Compute the FLP reduct — P^I
  - Delete from P every rule with a false body literal
- I is a stable model if I is a subset-minimal model of P^I
- $\mathcal{SM}(P)$: the set of all stable models of P

ASP Semantics: Example

$$
\begin{array}{rcl}
\text{“}NP \neq P\text{”} \vee \text{“}NP = P\text{”} & \leftarrow & \\
\text{“}NP = P\text{”} & \leftarrow & \text{“}polynomial\ algorithm\ for\ SAT\text{”} \\
\text{“}PH\ collapses\text{”} & \leftarrow & \text{“}NP = P\text{”} \\
\text{“}ASP\ harder\ than\ SAT\text{”} & \leftarrow & not\ \text{“}PH\ collapses\text{”}
\end{array}
$$

Stable models

1. $\{\text{“}NP \neq P\text{”}, \text{“}ASP\ harder\ than\ SAT\text{”}\}$
2. $\{\text{“}NP \neq P\text{”}, \text{“}NP = P\text{”}, \text{“}PH\ collapses\text{”}\}$
3. $\{\text{“}NP = P\text{”}, \text{“}PH\ collapses\text{”}\}$

- Compute the reduct, and
- Check minimality…

ASP Semantics: Example

$$
\begin{aligned}
\text{“}NP \neq P\text{”} \vee \text{“}NP = P\text{”} &\leftarrow \\
\text{“}NP = P\text{”} &\leftarrow \text{“polynomial algorithm for SAT”} \\
\text{“PH collapses”} &\leftarrow \text{“}NP = P\text{”} \\
\text{“ASP harder than SAT”} &\leftarrow \textit{not } \text{“PH collapses”}
\end{aligned}
$$

Stable models

1. $\{\text{“}NP \neq P\text{”}, \text{“ASP harder than SAT”}\}$
2. $\{\text{“}NP \neq P\text{”}, \text{“}NP = P\text{”}, \text{“PH collapses”}\}$
3. $\{\text{“}NP = P\text{”}, \text{“PH collapses”}\}$

- Compute the reduct, and
- Check minimality…

ASP Semantics: Example

$$\text{“}NP \neq P\text{”} \vee \text{“}NP = P\text{”} \leftarrow$$

$$\sout{\text{“}NP = P\text{”}} \leftarrow \sout{\text{“polynomial algorithm for SAT”}}$$

$$\sout{\text{“PH collapses”}} \leftarrow \sout{\text{“}NP = P\text{”}}$$

$$\text{“ASP harder than SAT”} \leftarrow \text{not “PH collapses”}$$

Stable models

1. $\{$“$NP \neq P$”, “ASP harder than SAT”$\}$
2. $\{$“$NP \neq P$”, “$NP = P$”, “PH collapses”$\}$
3. $\{$“$NP = P$”, “PH collapses”$\}$

- Compute the reduct, and
- Check minimality…

ASP Semantics: Example

$\text{“}NP \neq P\text{”} \vee \text{“}NP = P\text{”} \leftarrow$

~~$\text{“}NP = P\text{”}$~~ $\leftarrow$ ~~$\text{“}polynomial\ algorithm\ for\ SAT\text{”}$~~

~~$\text{“}PH\ collapses\text{”}$~~ $\leftarrow$ ~~$\text{“}NP = P\text{”}$~~

$\text{“}ASP\ harder\ than\ SAT\text{”} \leftarrow$ $not\ \text{“}PH\ collapses\text{”}$

Stable models

1. $\{\text{“}NP \neq P\text{”}, \text{“}ASP\ harder\ than\ SAT\text{”}\}$
2. $\{\text{“}NP \neq P\text{”}, \text{“}NP = P\text{”}, \text{“}PH\ collapses\text{”}\}$
3. $\{\text{“}NP = P\text{”}, \text{“}PH\ collapses\text{”}\}$

- Compute the reduct, and
- Check minimality...minimal!

ASP Semantics: Example

$$
\begin{array}{rcl}
\text{“}NP \neq P\text{”} \vee \text{“}NP = P\text{”} & \leftarrow & \\
\text{“}NP = P\text{”} & \leftarrow & \text{“polynomial algorithm for SAT”} \\
\text{“PH collapses”} & \leftarrow & \text{“}NP = P\text{”} \\
\text{“ASP harder than SAT”} & \leftarrow & \text{not “PH collapses”}
\end{array}
$$

Stable models

1. {“NP ≠ P”, “ASP harder than SAT”}
2. {“NP ≠ P”, “NP = P”, “PH collapses”}
3. {“NP = P”, “PH collapses”}

- Compute the reduct, and
- Check minimality…

ASP Semantics: Example

"NP ≠ P" ∨ "NP = P"	←	
~~"NP = P"~~	←	~~"polynomial algorithm for SAT"~~
"PH collapses"	←	"NP = P"
~~"ASP harder than SAT"~~	←	~~not "PH collapses"~~

Stable models

1. {"NP ≠ P", "ASP harder than SAT"}
2. {"NP ≠ P", "NP = P", "PH collapses"}
3. {"NP = P", "PH collapses"}

- Compute the reduct, and
- Check minimality...

ASP Semantics: Example

$$\text{“}NP \neq P\text{”} \vee \text{“}NP = P\text{”} \leftarrow$$
$$\text{~~“}NP = P\text{”~~} \leftarrow \text{~~“}polynomial\ algorithm\ for\ SAT\text{”~~}$$
$$\text{“}PH\ collapses\text{”} \leftarrow \text{“}NP = P\text{”}$$
$$\text{~~“}ASP\ harder\ than\ SAT\text{”~~} \leftarrow \text{~~}not\ \text{“}PH\ collapses\text{”~~}$$

Stable models

1. $\{$“$NP \neq P$”, “ASP $harder$ $than$ SAT”$\}$
2. ~~$\{$“$NP \neq P$”, “$NP = P$”, “PH $collapses$”$\}$~~
3. $\{$“$NP = P$”, “PH $collapses$”$\}$

- Compute the reduct, and
- Check minimality. . . countermodel: $\{$“$NP \neq P$”$\}$

ASP Semantics: Example

$$
\begin{aligned}
\text{“}NP \neq P\text{”} \vee \text{“}NP = P\text{”} &\leftarrow \\
\text{“}NP = P\text{”} &\leftarrow \text{“\textit{polynomial algorithm for SAT}”} \\
\text{“\textit{PH collapses}”} &\leftarrow \text{“}NP = P\text{”} \\
\text{“\textit{ASP harder than SAT}”} &\leftarrow \textit{not } \text{“\textit{PH collapses}”}
\end{aligned}
$$

Stable models

1. $\{\text{“}NP \neq P\text{”}, \text{“\textit{ASP harder than SAT}”}\}$
2. ~~$\{\text{“}NP \neq P\text{”}, \text{“}NP = P\text{”}, \text{“\textit{PH collapses}”}\}$~~
3. $\{\text{“}NP = P\text{”}, \text{“\textit{PH collapses}”}\}$

- Compute the reduct, and
- Check minimality…

ASP Semantics: Example

$$\text{“}NP \neq P\text{”} \vee \text{“}NP = P\text{”} \leftarrow$$
$$\text{~~“}NP = P\text{”~~} \leftarrow \text{~~“polynomial algorithm for SAT”~~}$$
$$\text{“PH collapses”} \leftarrow \text{“}NP = P\text{”}$$
$$\text{~~“ASP harder than SAT”~~} \leftarrow \text{~~not “PH collapses”~~}$$

Stable models

1. $\{\text{“}NP \neq P\text{”}, \text{“ASP harder than SAT”}\}$
2. ~~$\{\text{“}NP \neq P\text{”}, \text{“}NP = P\text{”}, \text{“PH collapses”}\}$~~
3. $\{\text{“}NP = P\text{”}, \text{“PH collapses”}\}$

- Compute the reduct, and
- Check minimality…

ASP Semantics: Example

$$
\begin{array}{rcl}
\text{“}NP \neq P\text{”} \vee \text{“}NP = P\text{”} & \leftarrow & \\
\sout{\text{“}NP = P\text{”}} & \leftarrow & \sout{\text{“}polynomial\ algorithm\ for\ SAT\text{”}} \\
\text{“}PH\ collapses\text{”} & \leftarrow & \text{“}NP = P\text{”} \\
\sout{\text{“}ASP\ harder\ than\ SAT\text{”}} & \leftarrow & \sout{not\ \text{“}PH\ collapses\text{”}}
\end{array}
$$

Stable models

1. $\{\text{“}NP \neq P\text{”}, \text{“}ASP\ harder\ than\ SAT\text{”}\}$
2. ~~$\{\text{“}NP \neq P\text{”}, \text{“}NP = P\text{”}, \text{“}PH\ collapses\text{”}\}$~~
3. $\{\text{“}NP = P\text{”}, \text{“}PH\ collapses\text{”}\}$

- Compute the reduct, and
- Check minimality…minimal!

Super-coherence Problems

Definition (Super-coherent Programs)

A program P is super-coherent if, for every set of facts F, the program $P \cup F$ is coherent, that is, $\mathcal{SM}(P \cup F) \neq \emptyset$.

We are interested in the complexity of the following decisional problems:

- Deciding super-coherence of disjunctive programs
- Deciding super-coherence of normal programs

Deciding Super-coherence (1)

Example 1

$a \vee b.$

Example 2

$a \leftarrow not\ b.$

Example 3

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

1. Positive programs are super-coherent
2. Stratified programs are super-coherent
3. Odd-cycle free programs are super-coherent

Deciding Super-coherence (1)

Example 1

$a \vee b.$

Example 2

$a \leftarrow not\ b.$

Example 3

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

A positive program

- Positive programs are coherent
- Adding facts cannot introduce negation

1. Positive programs are super-coherent
2. Stratified programs are super-coherent
3. Odd-cycle free programs are super-coherent

Deciding Super-coherence (1)

Example 1

$a \lor b.$

Example 2

$a \leftarrow not\ b.$

Example 3

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

A positive program

- Positive programs are coherent
- Adding facts cannot introduce negation

1. Positive programs are super-coherent
2. Stratified programs are super-coherent
3. Odd-cycle free programs are super-coherent

Deciding Super-coherence (1)

Example 1

$a \vee b.$

Example 2

$a \leftarrow not\ b.$

Example 3

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

1. Positive programs are super-coherent
2. Stratified programs are super-coherent
3. Odd-cycle free programs are super-coherent

Deciding Super-coherence (1)

Example 1

$a \vee b.$

Example 2

$a \leftarrow not\ b.$

Example 3

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

A stratified program

- Stratified programs are coherent
- Adding facts cannot introduce cycles

1. Positive programs are super-coherent
2. Stratified programs are super-coherent
3. Odd-cycle free programs are super-coherent

Deciding Super-coherence (1)

Example 1

$a \vee b.$

Example 2

$a \leftarrow not\ b.$

Example 3

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

A stratified program

- Stratified programs are coherent
- Adding facts cannot introduce cycles

1. Positive programs are super-coherent
2. Stratified programs are super-coherent
3. Odd-cycle free programs are super-coherent

Deciding Super-coherence (1)

Example 1

$a \vee b.$

Example 2

$a \leftarrow not\ b.$

Example 3

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

1. Positive programs are super-coherent
2. Stratified programs are super-coherent
3. Odd-cycle free programs are super-coherent

Deciding Super-coherence (1)

Example 1

$a \vee b.$

Example 2

$a \leftarrow not\ b.$

Example 3

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

An odd-cycle free program

- Odd-cycle free programs are coherent
- Adding facts cannot introduce new cycles

1. Positive programs are super-coherent
2. Stratified programs are super-coherent
3. Odd-cycle free programs are super-coherent

Deciding Super-coherence (1)

Example 1

$a \vee b.$

Example 2

$a \leftarrow not\ b.$

Example 3

$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

An odd-cycle free program

- Odd-cycle free programs are coherent
- Adding facts cannot introduce new cycles

1. Positive programs are super-coherent
2. Stratified programs are super-coherent
3. Odd-cycle free programs are super-coherent

Deciding Super-coherence (2)

- Up to odd-cycle free programs, it is a trivial problem
- The general case is not so easy!

Which of the programs is super-coherent?

$$P = \{ \quad a \leftarrow \qquad \leftarrow not\ b,\ not\ c \qquad c \leftarrow not\ b \quad \}$$

$$Q = \{ \quad a \leftarrow c \qquad b \vee c \leftarrow \qquad c \leftarrow not\ a \quad \}$$

We have:

- $\mathcal{SM}(P) = \mathcal{SM}(Q) = \{\{a, c\}\}$, but
- $\mathcal{SM}(P \cup \{b\}) = \{\{a, b\}\}$ and $\mathcal{SM}(Q \cup \{b\}) = \emptyset$.
- In fact, P is super-coherent!

Deciding Super-coherence (2)

- Up to odd-cycle free programs, it is a trivial problem
- The general case is not so easy!

Which of the programs is super-coherent?

$$P = \left\{ \begin{array}{lll} a & \leftarrow & \\ & \leftarrow & not\ b,\ not\ c \\ c & \leftarrow & not\ b \end{array} \right\} \qquad Q = \left\{ \begin{array}{rll} a & \leftarrow & c \\ b \lor c & \leftarrow & \\ c & \leftarrow & not\ a \end{array} \right\}$$

We have:

- $\mathcal{SM}(P) = \mathcal{SM}(Q) = \{\{a, c\}\}$, but
- $\mathcal{SM}(P \cup \{b\}) = \{\{a, b\}\}$ and $\mathcal{SM}(Q \cup \{b\}) = \emptyset$.
- In fact, P is super-coherent!

Deciding Super-coherence (2)

- Up to odd-cycle free programs, it is a trivial problem
- The general case is not so easy!

Which of the programs is super-coherent?

$$P = \left\{ \begin{array}{rcl} a & \leftarrow & \\ & \leftarrow & \textit{not } b,\ \textit{not } c \\ c & \leftarrow & \textit{not } b \end{array} \right\} \qquad Q = \left\{ \begin{array}{rcl} a & \leftarrow & c \\ b \vee c & \leftarrow & \\ c & \leftarrow & \textit{not } a \end{array} \right\}$$

We have:

- $\mathcal{SM}(P) = \mathcal{SM}(Q) = \{\{a, c\}\}$, but
- $\mathcal{SM}(P \cup \{b\}) = \{\{a, b\}\}$ and $\mathcal{SM}(Q \cup \{b\}) = \emptyset$.
- In fact, P is super-coherent!

Outline

1 Super-coherent ASP Programs
- Introduction, Motivation and Contribution
- Definitions and Examples

2 Main Results
- Proof Sketch
- Consequence of our results

Main Results

Theorem

The problem of deciding super-coherence for disjunctive programs is Π_3^P-complete.

Theorem

The problem of deciding super-coherence for normal programs is Π_2^P-complete.

Outline

1. Super-coherent ASP Programs
   - Introduction, Motivation and Contribution
   - Definitions and Examples
2. Main Results
   - Proof Sketch
   - Consequence of our results

Membership

Π_3^P-membership follows by the following algorithm for the complementary problem:

- guess a set $F \subseteq At(P)$ and check $\mathcal{SM}(P \cup F) = \emptyset$ via an oracle-call
- checking $\mathcal{SM}(P \cup F) = \emptyset$ is known to be in Π_2^P [Eiter, Gottlob; 95]

Hardness

Π_3^P-hardness is shown via a reduction from the evaluation problem of QBFs $\Phi = \forall X \exists Y \forall Z \phi$ to super-coherence of programs P_Φ in two steps:

1. we define required properties for P_Φ and show for programs satisfying these properties:

   Φ is true if and only if P_Φ is super-coherent

2. we provide a poly-time construction of P_Φ from Φ

Hardness — Step 1: Required Properties (1)

Definition (Φ-reduction)

Let $\Phi = \forall X \exists Y \forall Z \phi$ be a QBF with ϕ in DNF; call a program P satisfying the following properties a Φ-reduction:

1. P is given over atoms $U = X \cup Y \cup Z \cup \overline{X} \cup \overline{Y} \cup \overline{Z} \cup \{u, v, w\}$;
2. P has as its models: U and for each $I \subseteq X$, $J \subseteq Y$,

$$M[I, J] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup Z \cup \overline{Z} \cup \{v, u\}$$
$$M'[I, J] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup Z \cup \overline{Z} \cup \{v, w\};$$

3. models of $P^{M[I,J]}$ are $M[I, J]$ and $O[I] = I \cup \overline{(X \setminus I)}$;
4. models of $P^{M'[I,J]}$ are $M'[I, J]$ and $\forall K \subseteq Z$ s.t. $I \cup J \cup K \not\models \phi$,

$$N[I, J, K] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup K \cup \overline{(Z \setminus K)} \cup \{v\};$$

5. models of P^U are given only by the models mentioned above.

Hardness — Step 1: Required Properties (1)

Definition (Φ-reduction)

Let $\Phi = \forall X \exists Y \forall Z \phi$ be a QBF with ϕ in DNF; call a program P satisfying the following properties a Φ-reduction:

1. P is given over atoms $U = X \cup Y \cup Z \cup \overline{X} \cup \overline{Y} \cup \overline{Z} \cup \{u, v, w\}$;
2. P has as its models: U and for each $I \subseteq X$, $J \subseteq Y$,
$$M[I,J] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup Z \cup \overline{Z} \cup \{v, u\}$$
$$M'[I,J] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup Z \cup \overline{Z} \cup \{v, w\};$$
3. models of $P^{M[I,J]}$ are $M[I,J]$ and $O[I] = I \cup \overline{(X \setminus I)}$;
4. models of $P^{M'[I,J]}$ are $M'[I,J]$ and $\forall K \subseteq Z$ s.t. $I \cup J \cup K \not\models \phi$,
$$N[I,J,K] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup K \cup \overline{(Z \setminus K)} \cup \{v\};$$
5. models of P^U are given only by the models mentioned above.

Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.

Φ false

Φ true

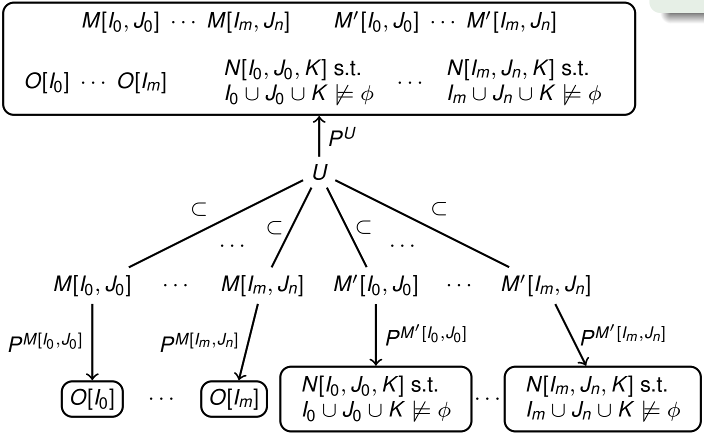
Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.

Φ false

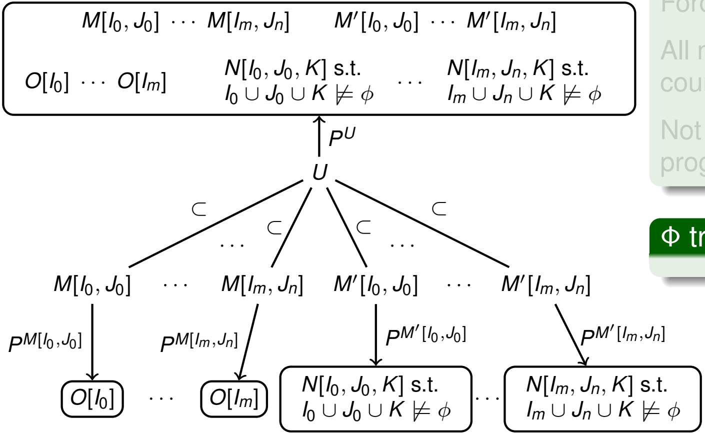
Φ true



Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.



Φ false

Let I_0 be s.t. $\forall Y \exists Z\ \phi(I_0)$ is false.

Force I_0 via facts.

All models have countermodels.

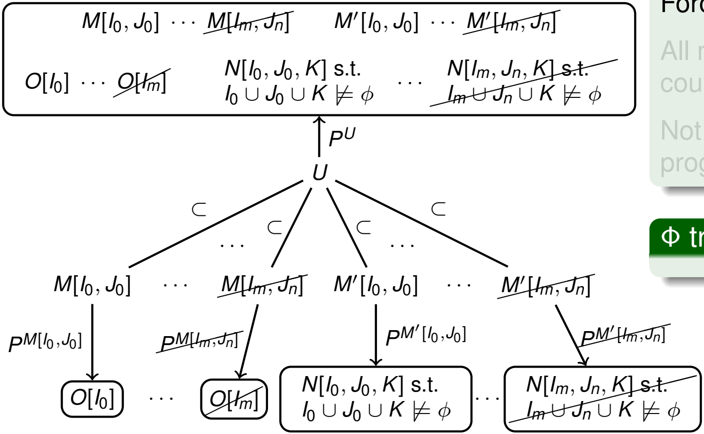
Not a super-coherent program!

Φ true

Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.



Φ false

Let I_0 be s.t. $\forall Y \exists Z\ \phi(I_0)$ is false.

Force I_0 via facts.

All models have countermodels.

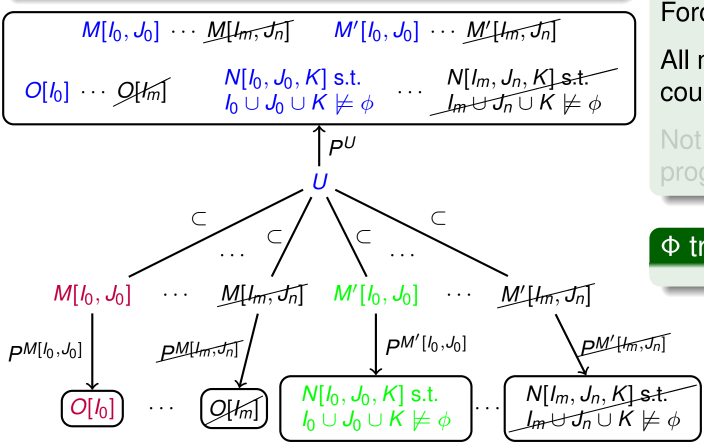
Not a super-coherent program!

Φ true

Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.



Φ false

Let I_0 be s.t. $\forall Y \exists Z\ \phi(I_0)$ is false.

Force I_0 via facts.

All models have countermodels.

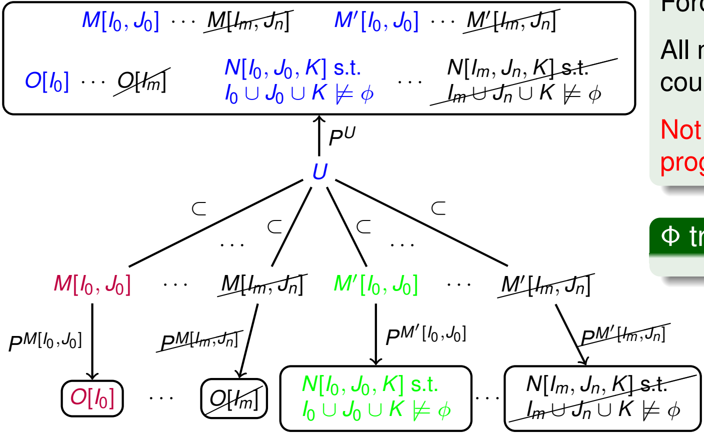
Not a super-coherent program!

Φ true

Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.



Φ false

Let I_0 be s.t. $\forall Y \exists Z\ \phi(I_0)$ is false.

Force I_0 via facts.

All models have countermodels.

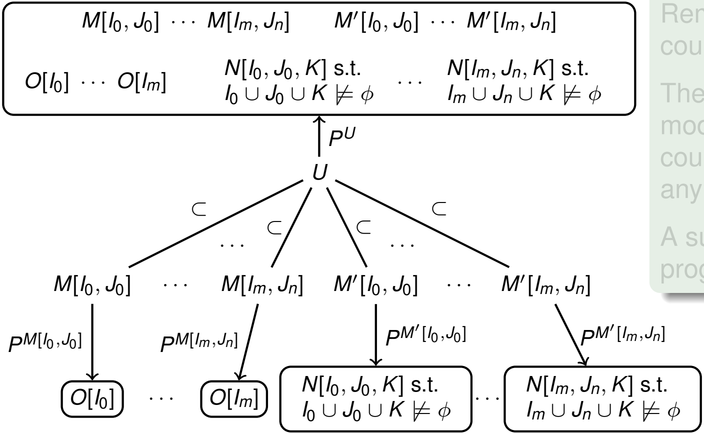
Not a super-coherent program!

Φ true

Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.



Φ false

Φ true

Remove inapplicable countermodels.

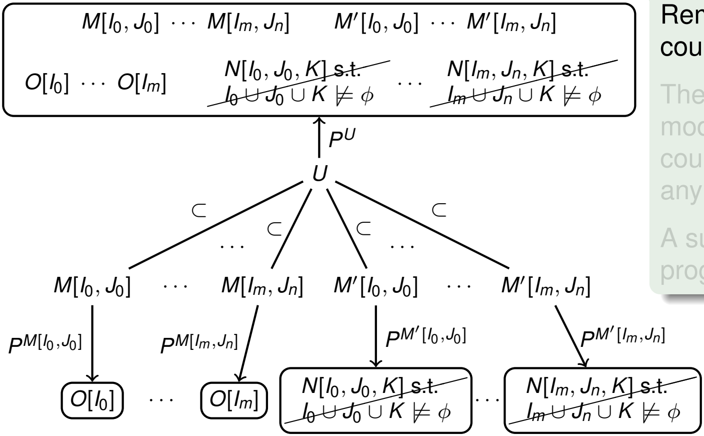
There is always a model with no countermodels (for any choice of facts).

A super-coherent program!

Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.



Φ false

Φ true

Remove inapplicable countermodels.

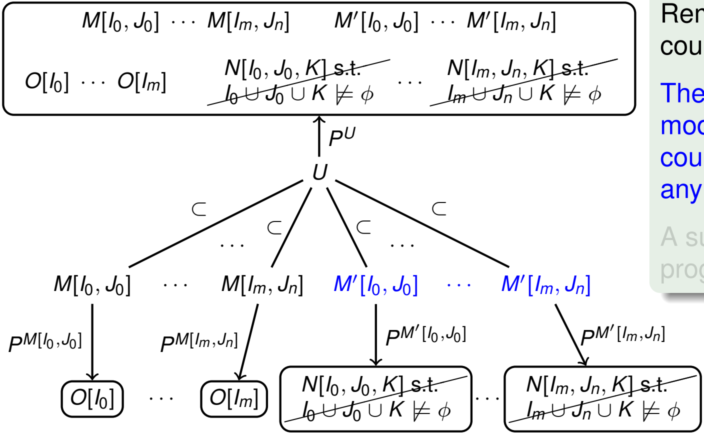
There is always a model with no countermodels (for any choice of facts).

A super-coherent program!

Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.



Φ false

Φ true

Remove inapplicable countermodels.

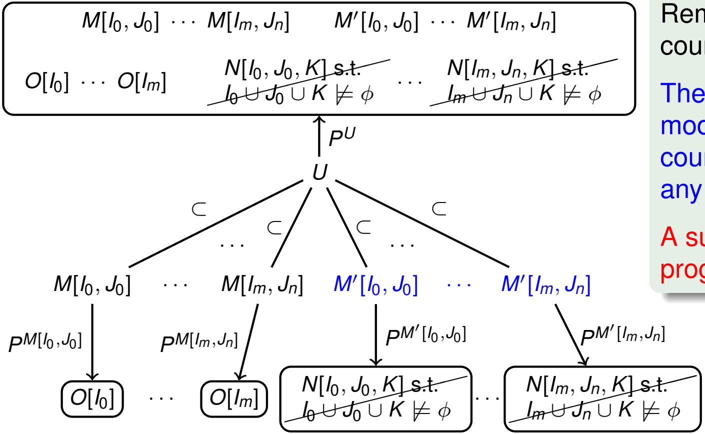
There is always a model with no countermodels (for any choice of facts).

A super-coherent program!

Hardness — Step 1: Required Properties (2)

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.



Φ false

Φ true

Remove inapplicable countermodels.

There is always a model with no countermodels (for any choice of facts).

A super-coherent program!

Hardness — Step 2: Poly-time Reduction

Definition

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with $\phi = \bigvee_{i=1}^{n} l_{i,1} \wedge \cdots \wedge l_{i,m_i}$ a DNF, define P_Φ as follows:

$$\{x \vee \overline{x} \leftarrow;\ u \leftarrow x, \overline{x};\ w \leftarrow x, \overline{x};\ x \leftarrow u, w;\ \overline{x} \leftarrow u, w \mid x \in X\} \cup$$
$$\{y \vee \overline{y} \leftarrow v;\ u \leftarrow y, \overline{y};\ w \leftarrow y, \overline{y};\ y \leftarrow u, w;$$
$$\overline{y} \leftarrow u, w;\ v \leftarrow y;\ v \leftarrow \overline{y} \mid y \in Y\} \cup$$
$$\{z \vee \overline{z} \leftarrow v;\ u \leftarrow z, not\ w;\ u \leftarrow \overline{z}, not\ w;\ v \leftarrow z;\ v \leftarrow \overline{z};$$
$$z \leftarrow w;\ \overline{z} \leftarrow w;\ z \leftarrow u;\ \overline{z} \leftarrow u;\ w \vee u \leftarrow z, \overline{z} \mid z \in Z\} \cup$$
$$\{w \vee u \leftarrow l_{i,1}, \ldots, l_{i,m_i} \mid 1 \leq i \leq n\}$$
$$\{v \leftarrow w;\ v \leftarrow u;\ v \leftarrow not\ u\}.$$

Lemma

For any QBF $\Phi = \forall X \exists Y \forall Z \phi$, the program P_Φ is a Φ-reduction.

Outline

1 Super-coherent ASP Programs
- Introduction, Motivation and Contribution
- Definitions and Examples

2 Main Results
- Proof Sketch
- Consequence of our results

Related Problem: Uniform Equivalence with Projection

Definition (Oetsch, Tompits, Woltran; 2007)

Given programs P and Q, and two sets A, B of atoms, $P \equiv^A_B Q$ if and only if, for each set $F \subseteq A$,

$$\{I \cap B \mid I \in \mathcal{SM}(P \cup F)\} = \{I \cap B \mid I \in \mathcal{SM}(Q \cup F)\}.$$

- Known: complexity of deciding $P \equiv^A_B Q$ is Π^P_3-complete for disjunctive programs;
  - however, hardness was only shown for bound context alphabets $A \subset U$
- Consequence of our results: $P \equiv^A_B Q$ remains Π^P_3-hard for $A = U$ and Q the empty program

Related Problem: Uniform Equivalence with Projection

Definition (Oetsch, Tompits, Woltran; 2007)

Given programs P and Q, and two sets A, B of atoms, $P \equiv^A_B Q$ if and only if, for each set $F \subseteq A$,

$$\{I \cap B \mid I \in \mathcal{SM}(P \cup F)\} = \{I \cap B \mid I \in \mathcal{SM}(Q \cup F)\}.$$

- Known: complexity of deciding $P \equiv^A_B Q$ is Π^P_3-complete for disjunctive programs;
  - however, hardness was only shown for bound context alphabets $A \subset U$
- Consequence of our results: $P \equiv^A_B Q$ remains Π^P_3-hard for $A = U$ and Q the empty program

Related Problem: Uniform Equivalence with Projection

Definition (Oetsch, Tompits, Woltran; 2007)

Given programs P and Q, and two sets A, B of atoms, $P \equiv_B^A Q$ if and only if, for each set $F \subseteq A$,

$$\{I \cap B \mid I \in \mathcal{SM}(P \cup F)\} = \{I \cap B \mid I \in \mathcal{SM}(Q \cup F)\}.$$

- Known: complexity of deciding $P \equiv_B^A Q$ is Π_3^P-complete for disjunctive programs;
  - however, hardness was only shown for bound context alphabets $A \subset U$
- Consequence of our results: $P \equiv_B^A Q$ remains Π_3^P-hard for $A = U$ and Q the empty program

Conclusion

Outline

- We studied the property of super-coherence; i.e. (here: propositional) programs which remain coherent no matter which facts are added
- Super-coherent programs have some nice properties and applications
- Complexity of deciding whether a program is super-coherent is rather high:
  - Π_3^P-complete for disjunctive programs
  - Π_2^P-complete for normal programs
- **Future Work:** Are there certain problems which become easier for super-coherent programs?

Conclusion

- We studied the property of super-coherence; i.e. (here: propositional) programs which remain coherent no matter which facts are added
- Super-coherent programs have some nice properties and applications
- Complexity of deciding whether a program is super-coherent is rather high:
  - Π_3^P-complete for disjunctive programs
  - Π_2^P-complete for normal programs
- **Future Work:** Are there certain problems which become easier for super-coherent programs?

Thank you!

Questions?

- We studied the property of super-coherence; i.e. (here: propositional) programs which remain coherent no matter which facts are added
- Super-coherent programs have some nice properties and applications
- Complexity of deciding whether a program is super-coherent is rather high:
  - Π_3^P-complete for disjunctive programs
  - Π_2^P-complete for normal programs
- **Future Work:** Are there certain problems which become easier for super-coherent programs?