

# Hyperequivalence in Logic Programming

Stefan Woltran  
*Technische Universität Wien*  
*A-1040 Vienna, Austria*

August 19, 2008

# Introduction

## Equivalence notions for knowledge-based formalisms

- Theoretical underpinning for
  - ▶ query rewriting/optimization
  - ▶ knowledge-base rewriting
- Query rewriting:  $Q \cup KB$  and  $Q' \cup KB$  have the same meaning, for *each* knowledge-base  $KB$
- KB rewriting; if  $KB$  has a modular structure: replace module  $M$  of  $KB$  by  $M'$  without changing the semantics of  $KB$ .
- Fine-grained eq-notions important for *application specific* rewritings.

## Observation

For classical logic, all is simple! (Replacement theorem)

# Introduction

## Equivalence notions for knowledge-based formalisms

- Theoretical underpinning for
  - ▶ query rewriting/optimization
  - ▶ knowledge-base rewriting
- Query rewriting:  $Q \cup KB$  and  $Q' \cup KB$  have the same meaning, for *each* knowledge-base  $KB$
- KB rewriting; if  $KB$  has a modular structure: replace module  $M$  of  $KB$  by  $M'$  without changing the semantics of  $KB$ .
- Fine-grained eq-notions important for *application specific* rewritings.

## Observation

For classical logic, all is simple! (Replacement theorem)

# Introduction (ctd.)

## The Answer-Set Programming Paradigm

- Basic Idea: Given a problem find an encoding (schema) to some formalism such that the **models** of the latter correspond to the **solutions** of the problem. Typical methods:
  - ▶ new reduction for each problem **instance**;
  - ▶ reduce **problem** to a single query, such that input + query solves problem;
- KB + Q can be formulated as logic program rules (not necessarily separated as in DB context)
- LP semantics are suitable for KR (transitive closure)
- Sophisticated LP engines available

## Introduction (ctd)

### Example: 3-Colorability

$$r(X) \vee g(X) \vee b(X) : \neg v(X).$$

$$\perp : \neg r(X), r(Y), e(X, Y).$$

$$\perp : \neg g(X), g(Y), e(X, Y).$$

$$\perp : \neg b(X), b(Y), e(X, Y).$$

# Road-map

- Motivation
  - ▶ Introduction of different equivalence notions
  - ▶ Some historical remarks (from the DATALOG world)
- Background
  - ▶ Syntax and semantics for logic programs
- Results for the propositional case
  - ▶ Supported model semantics
  - ▶ Stable model semantics
- Results for the stable non-ground case

# Motivation

## Example

$\{ \text{edge}(a, b) . \text{edge}(b, c) . \dots \}$   $KB$

$\left\{ \begin{array}{l} \text{path}(X, Y) :- \text{edge}(X, Y) . \\ \text{path}(X, Z) :- \text{edge}(X, Y) , \text{path}(Y, Z) . \end{array} \right\}$   $Q1$

$\left\{ \begin{array}{l} \text{path}(X, Y) :- \text{edge}(X, Y) . \\ \text{path}(X, Z) :- \text{path}(X, Y) , \text{path}(Y, Z) . \end{array} \right\}$   $Q2$

## Ordinary Equivalence (OE)

- Do  $Q1 \cup KB$  and  $Q2 \cup KB$  have the same output?

## More interesting problem: Query Equivalence (QE)

- Do  $Q1 \cup KB$  and  $Q2 \cup KB$  have the same output, for each KB (i.e., for any set of edges)?

# Motivation

## Example

$\{ \text{edge}(a, b) . \text{edge}(b, c) . \dots \}$   $KB$

$\left\{ \begin{array}{l} \text{path}(X, Y) :- \text{edge}(X, Y) . \\ \text{path}(X, Z) :- \text{edge}(X, Y) , \text{path}(Y, Z) . \end{array} \right\}$   $Q1$

$\left\{ \begin{array}{l} \text{path}(X, Y) :- \text{edge}(X, Y) . \\ \text{path}(X, Z) :- \text{path}(X, Y) , \text{path}(Y, Z) . \end{array} \right\}$   $Q2$

## Ordinary Equivalence (OE)

- Do  $Q1 \cup KB$  and  $Q2 \cup KB$  have the same output?

## More interesting problem: Query Equivalence (QE)

- Do  $Q1 \cup KB$  and  $Q2 \cup KB$  have the same output, for each KB (i.e., for any set of edges)?

# Motivation

## Example

$\{ \text{edge}(a, b) . \text{edge}(b, c) . \dots \}$  *KB*

$\left\{ \begin{array}{l} \text{path}(X, Y) :- \text{edge}(X, Y) . \\ \text{path}(X, Z) :- \text{edge}(X, Y) , \text{path}(Y, Z) . \end{array} \right\}$  *Q1*

$\left\{ \begin{array}{l} \text{path}(X, Y) :- \text{edge}(X, Y) . \\ \text{path}(X, Z) :- \text{path}(X, Y) , \text{path}(Y, Z) . \end{array} \right\}$  *Q2*

## More interesting problem: Query Equivalence (QE)

- Do  $Q1 \cup KB$  and  $Q2 \cup KB$  have the same output, for each *KB* (i.e., for any set of edges)?

## Different problem: Uniform Equivalence (UE)

- Do  $Q1 \cup KB$  and  $Q2 \cup KB$  have the same output, for any set of facts (i.e., also paths may be part of the input)?

# Motivation

## Example

$\{ \text{edge}(a, b) . \text{edge}(b, c) . \dots \}$   $KB$

$\left\{ \begin{array}{l} \text{path}(X, Y) :- \text{edge}(X, Y) . \\ \text{path}(X, Z) :- \text{edge}(X, Y) , \text{path}(Y, Z) . \end{array} \right\}$   $Q1$

$\left\{ \begin{array}{l} \text{path}(X, Y) :- \text{edge}(X, Y) . \\ \text{path}(X, Z) :- \text{path}(X, Y) , \text{path}(Y, Z) . \end{array} \right\}$   $Q2$

## More interesting problem: Query Equivalence (QE)

- Do  $Q1 \cup KB$  and  $Q2 \cup KB$  have the same output, for each  $KB$  (i.e., for any set of edges)?

## Different problem: Uniform Equivalence (UE)

- Do  $Q1 \cup KB$  and  $Q2 \cup KB$  have the same output, for any set of facts (i.e., also paths may be part of the input)?

# Some Remarks

## Results for DATALOG

- QE is undecidable (if domain is infinite) [Shmueli; SIGMOD 1987]
- UE is decidable [Sagiv, 1988]
- OE and UE are complete for EXPTIME [Eiter,Gottlob,Manilla; TODS 1997]
  - ▶ for bound predicate arities these problems are CONP-complete [Eiter,Faber,Fink,W; AMAI 2007]

## Observation

- Parameterizing predicates in input-DB (“context”) provides a uniform view:
  - ▶ *At-head* equivalence: UE
  - ▶ *EDB-head* equivalence (*EDB*: extensional predicates): QE
  - ▶  $\emptyset$ -head equivalence: OE.

# Some Remarks

## Results for DATALOG

- QE is undecidable (if domain is infinite) [Shmueli; SIGMOD 1987]
- UE is decidable [Sagiv, 1988]
- OE and UE are complete for EXPTIME [Eiter,Gottlob,Manilla; TODS 1997]
  - ▶ for bound predicate arities these problems are CONP-complete [Eiter,Faber,Fink,W; AMAI 2007]

## Observation

- Parameterizing predicates in input-DB (“context”) provides a uniform view:
  - ▶ *At*-head equivalence: UE
  - ▶ *EDB*-head equivalence (*EDB*: extensional predicates): QE
  - ▶  $\emptyset$ -head equivalence: OE.

## Motivation (ctd.)

### Further example

$$\left\{ \begin{array}{l} \text{edge}(Y, X) \text{ :- edge}(X, Y) . \\ \text{path}(X, Y) \text{ :- edge}(X, Y) . \\ \text{path}(X, Z) \text{ :- edge}(X, Y), \text{ path}(Y, Z) . \end{array} \right\} P1$$
$$\left\{ \begin{array}{l} \text{path}(X, Y) \text{ :- edge}(X, Y) . \\ \text{path}(X, Z) \text{ :- edge}(X, Y), \text{ path}(Y, Z) . \\ \text{path}(X, Y) \text{ :- path}(Y, X) . \end{array} \right\} P2$$
$$\left\{ \begin{array}{l} \text{edge}(a, b) . \text{ edge}(b, c) . \dots \\ \dots \text{ :- path}(X, Y) . \\ \dots \text{ :- } \dots \end{array} \right\} P$$

### Strong Equivalence (SE)

- Do  $P1 \cup P$  and  $P2 \cup P$  have the same output for any program  $P$ ?

## Motivation (ctd.)

### Further example

$$\left\{ \begin{array}{l} \text{edge}(Y, X) \text{ :- edge}(X, Y) . \\ \text{path}(X, Y) \text{ :- edge}(X, Y) . \\ \text{path}(X, Z) \text{ :- edge}(X, Y), \text{ path}(Y, Z) . \end{array} \right\} P1$$

$$\left\{ \begin{array}{l} \text{path}(X, Y) \text{ :- edge}(X, Y) . \\ \text{path}(X, Z) \text{ :- edge}(X, Y), \text{ path}(Y, Z) . \\ \text{path}(X, Y) \text{ :- path}(Y, X) . \end{array} \right\} P2$$

$$\left\{ \begin{array}{l} \text{edge}(a, b) . \text{ edge}(b, c) . \dots \\ \dots \text{ :- path}(X, Y) . \\ \dots \text{ :- } \dots \end{array} \right\} P$$

### Application specific equivalence

- Do  $P1 \cup P$  and  $P2 \cup P$  have the same output (for each  $P$ , where `edge` appears only in rule heads and `path` only in rule bodies)?

# Some Remarks

## Results for DATALOG

- UE and SE coincide (thus also SE decidable).
- this changes when (default) negation in rule bodies comes into play!

## Observation

- Parameterizing the head-predicates and body-predicates separately provides a uniform view of all equivalence notions introduced!
- We refer to these versions of equivalence collectively as **hyper equivalence**.

# Some Remarks

## Results for DATALOG

- UE and SE coincide (thus also SE decidable).
- this changes when (default) negation in rule bodies comes into play!

## Observation

- Parameterizing the head-predicates and body-predicates separately provides a uniform view of all equivalence notions introduced!
- We refer to these versions of equivalence collectively as **hyper equivalence**.

# Hyperequivalence

## Definition

- $\mathcal{HB}(A, B)$  – all programs  $P$  such that  $hd(P) \subseteq A$  and  $body(P) \subseteq B$
- Programs  $P$  and  $Q$  are **equivalent relative to  $\mathcal{HB}(A, B)$**  if for every program  $R \in \mathcal{HB}(A, B)$ ,  $P \cup R$  and  $Q \cup R$  have the same output

## Instantiations

- $(At, At)$ -equivalence = *strong equivalence*
- $(At, \emptyset)$ -equivalence = *uniform equivalence*
- $(EDB, \emptyset)$ -equivalence = *query equivalence*
- $(\emptyset, \emptyset)$ -equivalence = *ordinary equivalence*
- $(A, B)$ -equivalence = *“application specific equivalence”*
- $(At \setminus A, At \setminus A)$ -equivalence ... for *modules* with local predicates  $A$
- ...

# Hyperequivalence

## Definition

- $\mathcal{HB}(A, B)$  – all programs  $P$  such that  $hd(P) \subseteq A$  and  $body(P) \subseteq B$
- Programs  $P$  and  $Q$  are **equivalent relative to  $\mathcal{HB}(A, B)$**  if for every program  $R \in \mathcal{HB}(A, B)$ ,  $P \cup R$  and  $Q \cup R$  have the same output

## Instantiations

- $(At, At)$ -equivalence = *strong equivalence*
- $(At, \emptyset)$ -equivalence = *uniform equivalence*
- $(EDB, \emptyset)$ -equivalence = *query equivalence*
- $(\emptyset, \emptyset)$ -equivalence = *ordinary equivalence*
- $(A, B)$ -equivalence = *“application specific equivalence”*
- $(At \setminus A, At \setminus A)$ -equivalence ... for *modules* with local predicates  $A$
- ...

# Background

## Logic Programs

- We first consider *propositional programs*, i.e., set of rules of the form

$$a \leftarrow b_1, \dots, b_m, \mathbf{not} \ c_1, \dots, \mathbf{not} \ c_n,$$

where  $a_j$ ,  $b_j$  and  $c_j$  are atoms from a countable set  $At$

## Supported-Models Semantics

- Define the operator  $T_P(M) = \{hd(r) \mid r \in P \text{ and } M \models body(r)\}$ .
- $M$  is a **supported model** of a program  $P$  if  $T_P(M) = M$

## Stable-Models Semantics

- Define the reduct  $P^M = \{hd(r) \leftarrow body^+(r) \mid M \models body^-(r)\}$ .
- $M$  is a **stable model** of a program  $P$  if  $M$  is a minimal model of  $P^M$ .

# Background

## Logic Programs

- We first consider *propositional programs*, i.e., set of rules of the form

$$a \leftarrow b_1, \dots, b_m, \mathbf{not} c_1, \dots, \mathbf{not} c_n,$$

where  $a_i$ ,  $b_i$  and  $c_i$  are atoms from a countable set  $At$

## Supported-Models Semantics

- Define the operator  $T_P(M) = \{hd(r) \mid r \in P \text{ and } M \models body(r)\}$ .
- $M$  is a **supported model** of a program  $P$  if  $T_P(M) = M$

## Stable-Models Semantics

- Define the reduct  $P^M = \{hd(r) \leftarrow body^+(r) \mid M \models body^-(r)\}$ .
- $M$  is a **stable model** of a program  $P$  if  $M$  is a minimal model of  $P^M$ .

# Background

## Logic Programs

- We first consider *propositional programs*, i.e., set of rules of the form

$$a \leftarrow b_1, \dots, b_m, \mathbf{not} c_1, \dots, \mathbf{not} c_n,$$

where  $a_i$ ,  $b_i$  and  $c_i$  are atoms from a countable set  $At$

## Supported-Models Semantics

- Define the operator  $T_P(M) = \{hd(r) \mid r \in P \text{ and } M \models body(r)\}$ .
- $M$  is a **supported model** of a program  $P$  if  $T_P(M) = M$

## Stable-Models Semantics

- Define the reduct  $P^M = \{hd(r) \leftarrow body^+(r) \mid M \models body^-(r)\}$ .
- $M$  is a **stable model** of a program  $P$  if  $M$  is a minimal model of  $P^M$ .

# Examples

- Consider program  $P = \{a \leftarrow a\}$ .
  - ▶ We have  $T_P(\emptyset) = \emptyset$  and  $T_P(\{a\}) = \{a\}$ ; both  $\emptyset$  and  $\{a\}$  are supported models of  $P$ .
  - ▶ We have  $P^\emptyset = P^{\{a\}} = P$ . Since  $\emptyset \models P$ ,  $\emptyset$  is the only stable model of  $P$ .
- Consider program  $P = \{a \leftarrow a; a \leftarrow \text{not } a\}$ .
  - ▶ We have  $T_P(\emptyset) = T_P(\{a\}) = \{a\}$ . Thus,  $\{a\}$  is the only supported model of  $P$ .
  - ▶  $P^\emptyset = \{a \leftarrow a; a \leftarrow \}$  and  $\emptyset \not\models P^\emptyset$ ;  
 $P^{\{a\}} = \{a \leftarrow a\}$  and thus  $\emptyset \models P^{\{a\}}$ .  
So,  $P$  has no stable model.

## Proposition

Each supported model is also a stable model, but the converse direction is not true in general.

## Examples

- Consider program  $P = \{a \leftarrow a\}$ .
  - ▶ We have  $T_P(\emptyset) = \emptyset$  and  $T_P(\{a\}) = \{a\}$ ; both  $\emptyset$  and  $\{a\}$  are supported models of  $P$ .
  - ▶ We have  $P^\emptyset = P^{\{a\}} = P$ . Since  $\emptyset \models P$ ,  $\emptyset$  is the only stable model of  $P$ .
- Consider program  $P = \{a \leftarrow a; a \leftarrow \text{not } a\}$ .
  - ▶ We have  $T_P(\emptyset) = T_P(\{a\}) = \{a\}$ . Thus,  $\{a\}$  is the only supported model of  $P$ .
  - ▶  $P^\emptyset = \{a \leftarrow a; a \leftarrow \}$  and  $\emptyset \not\models P^\emptyset$ ;  
 $P^{\{a\}} = \{a \leftarrow a\}$  and thus  $\emptyset \models P^{\{a\}}$ .  
So,  $P$  has no stable model.

## Proposition

Each supported model is also a stable model, but the converse direction is not true in general.

# Equivalence in LP (Bibliography)

## Hyperequivalence for Stable Model Semantics

- SE [Lifschitz, Pearce, Valverde; ACM TOCL 2001]
- SE [Turner; TPLP 2003]
- UE [Eiter, Fink; ICLP 2003]
- relativized SE/UE [Eiter, Fink, W; ACM TOCL 2007]
- (“full”) HE: [W; TPLP 2008]
- relativized SE with proj.: [Eiter, Tompits, W; IJCAI 2005]
- relativized UE with proj.: [Oetsch, Tompits, W; AAAI 2007]

## Hyperequivalence for Supported Model Semantics

- [Truszczyński, W; AAAI 2008]

# Equivalence in LP (Bibliography)

## Hyperequivalence for Stable Model Semantics

- SE [Lifschitz, Pearce, Valverde; ACM TOCL 2001]
- SE [Turner; TPLP 2003]
- UE [Eiter, Fink; ICLP 2003]
- relativized SE/UE [Eiter, Fink, W; ACM TOCL 2007]
- (“full”) HE: [W; TPLP 2008]
- relativized SE with proj.: [Eiter, Tompits, W; IJCAI 2005]
- relativized UE with proj.: [Oetsch, Tompits, W; AAAI 2007]

## Hyperequivalence for Supported Model Semantics

- [Truszczyński, W; AAAI 2008]

# Supported Models

## Some Remarks

- Historically, the first semantics towards understanding negation in LP (Clark's completion);
- Approximation for (the more popular) stable models
  - ▶ “Stable models = Supported models + loop formulas”
- LP with supported models forms a significant fragment of the autoepistemic logic by Moore — one of the key formalisms of nonmonotonic reasoning.

# Strong Equivalence for Supported Models

## Definition

- $P$  and  $Q$  are *strongly supp-equivalent* if for every program  $R$ ,  $P \cup R$  and  $Q \cup R$  have the same supported models

## Candidate Models

- For a program  $P$  let  $Mod_{At}(P)$  denote the classical models of  $P$  (over  $At$ ).

## Theorem

- Let  $P$  and  $Q$  be programs. Then,  $P$  and  $Q$  are strongly supp-eq. iff  
 $Mod_{At}(P) = Mod_{At}(Q)$  and for any  $Y \in Mod_{At}(P)$ ,  $T_P(Y) = T_Q(Y)$ .

# Strong Equivalence for Supported Models

## Definition

- $P$  and  $Q$  are *strongly supp-equivalent* if for every program  $R$ ,  $P \cup R$  and  $Q \cup R$  have the same supported models

## Candidate Models

- For a program  $P$  let  $Mod_{At}(P)$  denote the classical models of  $P$  (over  $At$ ).

## Theorem

- Let  $P$  and  $Q$  be programs. Then,  $P$  and  $Q$  are strongly supp-eq. iff  
 $Mod_{At}(P) = Mod_{At}(Q)$  and for any  $Y \in Mod_{At}(P)$ ,  $T_P(Y) = T_Q(Y)$ .

# Strong Equivalence for Supported Models

## Definition

- $P$  and  $Q$  are *strongly supp-equivalent* if for every program  $R$ ,  $P \cup R$  and  $Q \cup R$  have the same supported models

## Candidate Models

- For a program  $P$  let  $Mod_{At}(P)$  denote the classical models of  $P$  (over  $At$ ).

## Theorem

- Let  $P$  and  $Q$  be programs. Then,  $P$  and  $Q$  are strongly supp-eq. iff  
 $Mod_{At}(P) = Mod_{At}(Q)$  and for any  $Y \in Mod_{At}(P)$ ,  $T_P(Y) = T_Q(Y)$ .

# Hyperequivalence for Supported Models

## Definition

- Let  $A, B \subseteq At$ .  $P$  and  $Q$  are *supp-equivalent relative to  $\mathcal{HB}(A, B)$*  if for every program  $R \in \mathcal{HB}(A, B)$ ,  $P \cup R$  and  $Q \cup R$  have the same supported models

## Candidate Models

- For a program  $P$  and  $A \subseteq At$ , define:  
$$Mod_A(P) = \{Y \subseteq At \mid Y \models P \text{ and } Y \setminus T_P(Y) \subseteq A\}.$$

## Theorem [Truszczyński, W: AAI 2008]

- Let  $P$  and  $Q$  be programs and  $A \subseteq At$ . Then,  
 $P$  and  $Q$  are supp-eq. relative to  $\mathcal{HB}(A, B)$  (for any  $\emptyset \subseteq B \subseteq At$ )  
iff  
$$Mod_A(P) = Mod_A(Q) \text{ and for any } Y \in Mod_A(P), T_P(Y) = T_Q(Y).$$

# Hyperequivalence for Supported Models

## Definition

- Let  $A, B \subseteq At$ .  $P$  and  $Q$  are *supp-equivalent relative to  $\mathcal{HB}(A, B)$*  if for every program  $R \in \mathcal{HB}(A, B)$ ,  $P \cup R$  and  $Q \cup R$  have the same supported models

## Candidate Models

- For a program  $P$  and  $A \subseteq At$ , define:  
 $Mod_A(P) = \{Y \subseteq At \mid Y \models P \text{ and } Y \setminus T_P(Y) \subseteq A\}$ .

## Theorem [Truszczyński, W: AAI 2008]

- Let  $P$  and  $Q$  be programs and  $A \subseteq At$ . Then,  
 $P$  and  $Q$  are supp-eq. relative to  $\mathcal{HB}(A, B)$  (for any  $\emptyset \subseteq B \subseteq At$ )  
iff  
 $Mod_A(P) = Mod_A(Q)$  and for any  $Y \in Mod_A(P)$ ,  $T_P(Y) = T_Q(Y)$ .

# Hyperequivalence for Supported Models

## Definition

- Let  $A, B \subseteq At$ .  $P$  and  $Q$  are *supp-equivalent relative to  $\mathcal{HB}(A, B)$*  if for every program  $R \in \mathcal{HB}(A, B)$ ,  $P \cup R$  and  $Q \cup R$  have the same supported models

## Candidate Models

- For a program  $P$  and  $A \subseteq At$ , define:  
 $Mod_A(P) = \{Y \subseteq At \mid Y \models P \text{ and } Y \setminus T_P(Y) \subseteq A\}$ .

## Theorem [Truszczyński, W: AAI 2008]

- Let  $P$  and  $Q$  be programs and  $A \subseteq At$ . Then,  
 $P$  and  $Q$  are supp-eq. relative to  $\mathcal{HB}(A, B)$  (for any  $\emptyset \subseteq B \subseteq At$ )  
iff  
 $Mod_A(P) = Mod_A(Q)$  and for any  $Y \in Mod_A(P)$ ,  $T_P(Y) = T_Q(Y)$ .

# Hyperequivalence for Supported Models (ctd.)

## Corollary

- The following conditions are equivalent:
  - ▶  $P$  and  $Q$  are strongly supp-equivalent (i.e., relative to  $\mathcal{HB}(At, At)$ ),
  - ▶  $P$  and  $Q$  are uniformly supp-equivalent (i.e., relative to  $\mathcal{HB}(At, \emptyset)$ ),
  - ▶  $P$  and  $Q$  have the same classical models and for every model  $Y$ , and  $T_P(Y) = T_Q(Y)$ .
- Hence no difference between SE and UE.
- This changes if one considers supported **minimal** models
  - ▶ But SE for supported models is the same as SE of supported minimal models
  - ▶ Hyperequivalence for supported minimal models has also been studied in [Truszczyński, W: AAI 2008]

## Hyperequivalence for Supported Models (ctd.)

### Example

- Is  $P = \{a \leftarrow a\}$  strongly supp-equivalent to the empty program?
- No!  $P = \{a \leftarrow a\}$  and  $\emptyset$  have the same classical models, but

$$T_P(\{a\}) = \{a\} \neq \emptyset = T_\emptyset(\{a\})$$

- Consider  $P = \{a \leftarrow a; a \leftarrow \mathbf{not} a\}$  and  $Q = \{a \leftarrow\}$ .
- $P$  and  $Q$  have the same classical models  $\{a\} \subseteq Y \subseteq At$ .
- For each such  $Y$ ,

$$T_P(Y) = \{a\} = T_Q(Y)$$

## Hyperequivalence for Supported Models (ctd.)

### Example

- Is  $P = \{a \leftarrow a\}$  strongly supp-equivalent to the empty program?
- No!  $P = \{a \leftarrow a\}$  and  $\emptyset$  have the same classical models, but

$$T_P(\{a\}) = \{a\} \neq \emptyset = T_\emptyset(\{a\})$$

- Consider  $P = \{a \leftarrow a; a \leftarrow \mathbf{not} a\}$  and  $Q = \{a \leftarrow\}$ .
- $P$  and  $Q$  have the same classical models  $\{a\} \subseteq Y \subseteq \mathbf{At}$ .
- For each such  $Y$ ,

$$T_P(Y) = \{a\} = T_Q(Y)$$

# Hyperequivalence for Supported Models (ctd.)

## Complexity

- The problem of hyperequivalence for supported models between normal programs is CONP-complete.
  - ▶ Results holds even for fixed alphabets and for disjunctive programs.
- Also the following problem is CONP-hard (for non-empty  $A$ ):
  - ▶ Given programs  $P, Q$ , such that  $Mod_A(P) = Mod_A(Q)$  holds; decided whether  $P$  and  $Q$  are equivalent wrt  $\mathcal{HB}(A, B)$ .

## Supported Minimal Models

- Complexity is  $\Pi_2^P$ -complete, even for normal programs.
- UE and SE remain CONP-complete, even for disj. programs.

# Stable Models

## Some Remarks

- Introduced by [Gelfond, Lifschitz: ICLP 1988, NGC 1991]
- Roots in Nonmonotonic Reasoning (“default negation”), but also generalizes DATALOG
- Prototypical formalism in ASP-paradigm
- Since mid 90ies: Development of efficient solvers (first competition at LPNMR 2007)
- Numerous applications and language extensions
  - ▶ IST project IST-FET-2001-37004 Working Group on Answer Set Programming ([WASP](#)); see eg:  
<http://www.kr.tuwien.ac.at/research/projects/WASP/>

# Equivalence Notions for Stable Models

## Observation

- Strong and uniform equivalence are different concepts:
- Consider  $P = \{a \leftarrow b; a \leftarrow \mathbf{not} b\}$  and  $Q = \{a \leftarrow c; a \leftarrow \mathbf{not} c\}$ .
- Quite intuitively, for each set  $F$  of facts,

$$SM(P \cup F) = SM(Q \cup F) = \{a\} \cup F$$

Thus  $P$  and  $Q$  are uniformly stable equivalent.

- But, we have

$$SM(P \cup \{b \leftarrow a\}) = \emptyset \quad SM(Q \cup \{b \leftarrow a\}) = \{\{a, b\}\}$$

Thus  $P$  and  $Q$  are *not* strongly stable equivalent.

# Strong Equivalence for Stable Models

## Definition

- $P$  and  $Q$  are *strongly stable equivalent* if for every program  $R$ ,  $P \cup R$  and  $Q \cup R$  have the same stable models

## Candidate Models

- Again are all classical models,  $Mod^{At}(P) = \{Y \mid Y \models P\}$

## Theorem [Turner; TPLP 2003]

- Programs  $P$  and  $Q$  are strongly stable equivalent iff  
 $Mod^{At}(P) = Mod^{At}(Q)$  and for any  $Y \in Mod^{At}(P)$  and any  $X \subseteq Y$ ,  $X \models P^Y$  iff  $X \models Q^Y$ .

## SE-models due to Turner

- pairs  $(X, Y)$  with  $X \subseteq Y$ ,  $Y \models P$ ,  $X \models P^Y$

# Strong Equivalence for Stable Models

## Definition

- $P$  and  $Q$  are *strongly stable equivalent* if for every program  $R$ ,  $P \cup R$  and  $Q \cup R$  have the same stable models

## Candidate Models

- Again are all classical models,  $Mod^{At}(P) = \{Y \mid Y \models P\}$

## Theorem [Turner; TPLP 2003]

- Programs  $P$  and  $Q$  are strongly stable equivalent iff  
 $Mod^{At}(P) = Mod^{At}(Q)$  and for any  $Y \in Mod^{At}(P)$  and any  $X \subseteq Y$ ,  $X \models P^Y$  iff  $X \models Q^Y$ .

## SE-models due to Turner

- pairs  $(X, Y)$  with  $X \subseteq Y$ ,  $Y \models P$ ,  $X \models P^Y$

# Strong Equivalence for Stable Models

## Definition

- $P$  and  $Q$  are *strongly stable equivalent* if for every program  $R$ ,  $P \cup R$  and  $Q \cup R$  have the same stable models

## Candidate Models

- Again are all classical models,  $Mod^{At}(P) = \{Y \mid Y \models P\}$

## Theorem [Turner; TPLP 2003]

- Programs  $P$  and  $Q$  are strongly stable equivalent iff  
 $Mod^{At}(P) = Mod^{At}(Q)$  and for any  $Y \in Mod^{At}(P)$  and any  $X \subseteq Y$ ,  $X \models P^Y$  iff  $X \models Q^Y$ .

## SE-models due to Turner

- pairs  $(X, Y)$  with  $X \subseteq Y$ ,  $Y \models P$ ,  $X \models P^Y$

# Hyperequivalence for Stable Models

## Definition

- $P$  and  $Q$  are *stable equivalent relative to*  $\mathcal{HB}(A, B)$  if for every program  $R$ ,  $R \in \mathcal{HB}(A, B)$ ,  $P \cup R$  and  $Q \cup R$  have the same stable models

## Candidate Models

- For a program  $P$  and  $A \subseteq At$ , define:  
$$Mod^A(P) = \{Y \subseteq At \mid Y \models P; \forall Y' \subset Y : Y' \models P^Y \Rightarrow Y'|_A \subset Y|_A\}.$$

## Theorem [W; TPLP 2008]

- Let  $A, B \subseteq At$ . Programs  $P$  and  $Q$  are stable eq. rel. to  $\mathcal{HB}(A, B)$  iff  
 $Mod^A(P) = Mod^A(Q)$  and for any  $Y \in Mod^A(P)$  and any  $X \subset Y$ ,
  - ▶ If  $X \models P^Y$  then there exists a  $Z \subset Y$  with  $X|_A \subseteq Z|_A$ ,  $Z|_B \subseteq X|_B$ , such that  $Z \models Q^Y$
  - ▶ If  $X \models Q^Y$  then there exists a  $Z \subset Y$  with  $X|_A \subseteq Z|_A$ ,  $Z|_B \subseteq X|_B$ , such that  $Z \models P^Y$

# Hyperequivalence for Stable Models

## Definition

- $P$  and  $Q$  are *stable equivalent relative to  $\mathcal{HB}(A, B)$*  if for every program  $R$ ,  $R \in \mathcal{HB}(A, B)$ ,  $P \cup R$  and  $Q \cup R$  have the same stable models

## Candidate Models

- For a program  $P$  and  $A \subseteq At$ , define:  
$$Mod^A(P) = \{Y \subseteq At \mid Y \models P; \forall Y' \subset Y : Y' \models P^Y \Rightarrow Y'|_A \subset Y|_A\}.$$

## Theorem [W; TPLP 2008]

- Let  $A, B \subseteq At$ . Programs  $P$  and  $Q$  are stable eq. rel. to  $\mathcal{HB}(A, B)$  iff  
 $Mod^A(P) = Mod^A(Q)$  and for any  $Y \in Mod^A(P)$  and any  $X \subset Y$ ,
  - ▶ If  $X \models P^Y$  then there exists a  $Z \subset Y$  with  $X|_A \subseteq Z|_A$ ,  $Z|_B \subseteq X|_B$ , such that  $Z \models Q^Y$
  - ▶ If  $X \models Q^Y$  then there exists a  $Z \subset Y$  with  $X|_A \subseteq Z|_A$ ,  $Z|_B \subseteq X|_B$ , such that  $Z \models P^Y$

# Hyperequivalence for Stable Models

## Definition

- $P$  and  $Q$  are *stable equivalent relative to*  $\mathcal{HB}(A, B)$  if for every program  $R$ ,  $R \in \mathcal{HB}(A, B)$ ,  $P \cup R$  and  $Q \cup R$  have the same stable models

## Candidate Models

- For a program  $P$  and  $A \subseteq At$ , define:  
$$Mod^A(P) = \{Y \subseteq At \mid Y \models P; \forall Y' \subset Y : Y' \models P^Y \Rightarrow Y'|_A \subset Y|_A\}.$$

## Theorem [W; TPLP 2008]

- Let  $A, B \subseteq At$ . Programs  $P$  and  $Q$  are stable eq. rel. to  $\mathcal{HB}(A, B)$  iff  
 $Mod^A(P) = Mod^A(Q)$  and for any  $Y \in Mod^A(P)$  and any  $X \subset Y$ ,
  - ▶ If  $X \models P^Y$  then there exists a  $Z \subset Y$  with  $X|_A \subseteq Z|_A$ ,  $Z|_B \subseteq X|_B$ , such that  $Z \models Q^Y$
  - ▶ If  $X \models Q^Y$  then there exists a  $Z \subset Y$  with  $X|_A \subseteq Z|_A$ ,  $Z|_B \subseteq X|_B$ , such that  $Z \models P^Y$

# Hyperequivalence for Stable Models (ctd.)

## Example

- Recall  $P = \{a \leftarrow b; a \leftarrow \text{not } b\}$  and  $Q = \{a \leftarrow c; a \leftarrow \text{not } c\}$ .

## Strong Equivalence

- $P$  and  $Q$  are classically equivalent, but for model  $\{a, b\}$   
 $\emptyset \models P^{\{a,b\}} = \{a \leftarrow b\}$  while  $\emptyset \not\models Q^{\{a,b\}} = \{a \leftarrow c; a \leftarrow \}$ .

## Uniform Equivalence

- $P$  and  $Q$  are UE iff they are classically equivalent, and for each model  $Y$ , and each  $X \subset Y$ ,
  - If  $X \models P^Y$  then there exists a  $Z \subset Y$  with  $X \subseteq Z$ , such that  $Z \models Q^Y$
  - If  $X \models Q^Y$  then there exists a  $Z \subset Y$  with  $X \subseteq Z$ , such that  $Z \models P^Y$
- One can check that this holds!

# Hyper-equivalence for Stable Models (ctd.)

## Complexity [Eiter, Fink, W; ACM TOCL 2007]

- The problem of hyper-equivalence for stable models between normal programs is CONP-complete and  $\Pi_2^P$ -complete for disjunctive programs.
  - ▶ Results holds even for fixed (finite) alphabets
  - ▶  $\Pi_2^P$  holds also for UE, but SE is CONP-complete also for disj. programs.
  - ▶ For Horn programs SE and UE are tractable, but HE in general remains CONP-complete
- Also the following problem is CONP-hard:
  - ▶ Given stratified programs  $P$ ,  $Q$ , decide whether  $P$  and  $Q$  are strongly equivalent.

# Hyperequivalence for Stable Models (ctd.)

Theorem [Eiter, Fink, Tompits, W; IJCAI 2007]

- Deciding SE between stratified programs is CONP-complete.
  - use an embedding of the NP-complete *exact hitting set* problem, given by  $(C, S)$ , where  $C$  is a family of subsets of a finite set  $S$ . The problem is to decide whether there exists a subset  $S' \subseteq S$  such that  $|S' \cap C'| = 1$ , for each  $C' \in C$ . Any such  $S'$  is an exact hitting set (EHS) for  $C$ . It is sufficient to consider  $C$  containing sets of cardinality 3. So, let  $C = \{\{c_{1,1}, c_{1,2}, c_{1,3}\}, \dots, \{c_{n,1}, c_{n,2}, c_{n,3}\}\}$ .
  - Given  $(C, S)$ , we construct stratified programs  $P$  and  $Q$  as follows.

$$R = \bigcup_{1 \leq i \leq n} \{ a \leftarrow c_{i,1}, c_{i,2}; a \leftarrow c_{i,1}, c_{i,3}; a \leftarrow c_{i,2}, c_{i,3}; \\ a \leftarrow \mathbf{not} c_{i,1}, \mathbf{not} c_{i,2}, \mathbf{not} c_{i,3}; \\ b \leftarrow c_{i,1}, c_{i,2}; b \leftarrow c_{i,1}, c_{i,3}; b \leftarrow c_{i,2}, c_{i,3}; \\ b \leftarrow \mathbf{not} c_{i,1}, \mathbf{not} c_{i,2}, \mathbf{not} c_{i,3} \},$$

$$P = R \cup \{ b \leftarrow \mathbf{not} a \}, \text{ and}$$

$$Q = R \cup \{ a \leftarrow \mathbf{not} b \}.$$

- It can be shown, that there exists an EHS for  $(C, S)$  iff  $P$  and  $Q$  are not strongly eq.

# Eq. for Stable Models — The Non-Ground Case

## Remarks

- Programs with predicates and **variables** can be viewed as abbreviation of propositional programs (remain finite if the domain is finite).
- For “complete” programs it is sufficient to use the “active domain”, i.e., the constants in the program.
- For equivalence tests, this is in general not possible (context may bring additional constants into play!)
  - ▶ Recall: QE is undecidable already for Horn programs

# Stable Models — The Non-Ground Case

## Definitions

- Let  $U_P \subseteq U$  be the active domain of program  $P$ .
- Let  $C \subseteq U$ ; the **grounding** of a program  $P$ ,  $G(P, C)$  is the union of all programs  $P\theta$  where  $\theta$  is any substitution of the variables in  $P$  by elements of  $C$ .
- A set  $I$  of ground atoms is a stable model of a program  $P$  iff  $I$  is a stable model of the ground program  $G(P, U_P)$ .

# Strong Equivalence— The Non-Ground Case

**Theorem** [Eiter, Fink, Tompits, W; AAI 2005]

- Given  $P$ , define the **extended** active domain as  $U_P^+ = U_P \cup \{c_1, \dots, c_n\}$ , where  $c_i$  are new constants from  $U$  and  $n$  is the max. number of variables in a rule of  $P$ .
- Then,  $P$  and  $Q$  are strongly equivalent iff  $G(P, U_{P \cup Q}^+)$  is strongly equivalent to  $G(Q, U_{P \cup Q}^+)$ .

## Consequence

- Strong Equivalence between Non-Ground programs remains decidable
- In fact, the problem is CONEXPTIME-complete.
- If predicate arity is bounded, the problem is  $\Pi_2^P$ -complete

[Eiter, Faber, Fink, W; AMAI 2007]

# Strong Equivalence— The Non-Ground Case

**Theorem** [Eiter, Fink, Tompits, W; AAI 2005]

- Given  $P$ , define the **extended** active domain as  $U_P^+ = U_P \cup \{c_1, \dots, c_n\}$ , where  $c_i$  are new constants from  $U$  and  $n$  is the max. number of variables in a rule of  $P$ .
- Then,  $P$  and  $Q$  are strongly equivalent iff  $G(P, U_{P \cup Q}^+)$  is strongly equivalent to  $G(Q, U_{P \cup Q}^+)$ .

## Consequence

- Strong Equivalence between Non-Ground programs remains decidable
- In fact, the problem is CONEXPTIME-complete.
- If predicate arity is bounded, the problem is  $\Pi_2^P$ -complete  
[Eiter, Faber, Fink, W; AAI 2007]

# Uniform Equivalence — The Non-Ground Case

## Theorem [Eiter,Fink,Tompits,W; AAI 2005, IJCAI 2007]

- Uniform equivalence is undecidable for normal programs.
  - ▶ Shown via a mapping of QE between Horn programs to UE between normal programs
- Undecidability holds already for “small” programs or programs with bounded predicate arity
- However, still some open problems concerning the decidability/undecidability frontier

## The Non-Ground Case — Further Remarks

- SE has been implemented [Eiter,Faber,Traxler; LPNMR 2005] via reduction to FO-logic fragment (as suggested by [Lin; KR 2002]).
- Logical characterization (circumventing grounding)  
[Lifschitz,Pearce,Valverde; LPNMR 2007]
- Applications [de Bruijn,Pearce,Polleres,Valverde; RR 2007]
- Towards hyperequivalence for non-ground programs [Oetsch,Tompits; ICLP 2008]

# Conclusions

- Equivalence in Logic Programming has received increasing interest in recent years;
- In this talk: Overview on different equivalence notions, characterizations, and complexity of the problem.
- Further related work is mainly concerned with
  - ▶ program transforms (work by our group; Lin and Chen; Wong; ...)
  - ▶ modular programming (work by Janhunen et al)
- Project: **Formal Methods for Comparing and Optimizing Nonmonotonic Logic Programs** (supported by the Austrian Science Foundation under grant P18019-N04.)  
<http://www.kr.tuwien.ac.at/research/projects/eq>