Synthesizing Strongly Equivalent Logic Programs: Beth Definability for Answer Set Programs via Craig Interpolation in First-Order Logic

Jan Heuer and Christoph Wernhard

University of Potsdam

IJCAR 2024

Nancy, July 3, 2024

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 457292495.

Beth Definability and Craig Interpolation in a Nutshell

Definition. Q is ***implicitly definable*** in terms of vocabulary V within K iff

$$K \land K' \models Q \leftrightarrow Q',$$

where K' and Q' are copies of K and Q with all symbols not in V replaced by fresh symbols

This says: If two models of K agree on values of symbols in V, then they agree on the value of Q

Definition. Q is ***explicitly definable*** in terms of vocabulary V within K iff

there exists a formula R in vocabulary V s.th. $K \models Q \leftrightarrow R$

[Beth 1953] In first-order logic implicit and explicit definability are equivalent

Definition. A ***Craig interpolant*** of F and G s.th. $F \models G$ is a formula H s.th.
(1.) $F \models H$ (2.) $H \models G$ (3.) The vocabulary of H is in the common vocabulary of F and G

[Craig 1957] In first-order logic H exists and can be extracted from a proof of $F \models G$

Proof of [Beth] via [Craig].
Write implicit definability as $K \land Q \models K' \rightarrow Q'$
Obtain R as Craig interpolant of $K \land Q$ and $K' \rightarrow Q'$

$$K \models Q \leftrightarrow R$$

$$K \models Q \rightarrow R \qquad K \models R \rightarrow Q$$

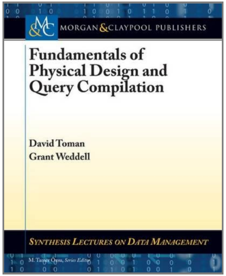
$$K \land Q \quad \models R \models \quad K' \rightarrow Q'$$

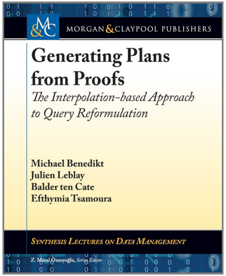
Beth via Craig in Databases and Knowledge Representation

$$K \models Q \leftrightarrow R$$

$$K \land Q \ \models \ R \ \models \ K' \rightarrow Q'$$

- Synthesis of definitions by Craig interpolation is a logic-based technique for **query reformulation**





[Nash/Segoufin/Vianu 2005, 2010]
[Toman/Wedell 2011]
[Benedikt et al. 2016]

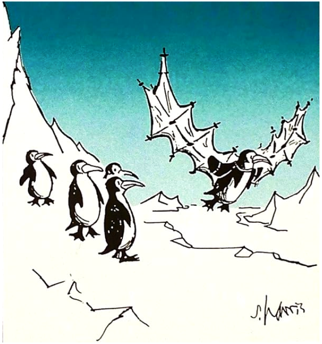
- Strengthened variations of Craig interpolation preserve criteria for domain independence, e.g., through **relativized quantifiers** [Benedikt et al. 2015] or **range-restriction** [W 2023]

Our Question: Beth via Craig to Synthesize Answer Set Programs?

$$P \models Q \leftrightarrow R$$

$$P \land Q \models R \models P' \rightarrow Q'$$

- Idea: For given logic programs P, Q and vocabulary V **synthesize** a program R **in** V that is **"equivalent"** to Q under assumptions P
- But: Logic programs are considered under **nonmonotonic semantics**



From the cover of Matthew L. Ginsberg (ed.): Readings in Nonmonotonic Reasoning, 1987

- A ***logic program*** is a set of ***rules*** of the form

$$A_1; \ldots; A_k; \textbf{not } A_{k+1}; \ldots; \textbf{not } A_l \leftarrow A_{l+1}, \ldots, A_m, \textbf{not } A_{m+1}, \ldots, \textbf{not } A_n$$

  - I.e., we consider disjunctive logic programs with negation in the head
  - Atoms can have argument terms built from variables, constants and function symbols
- An **answer set solver** computes **answer sets** (stable models [Gelfond/Lifschitz 1988]) of a program
  - These are minimal Herbrand models in which all facts are properly justified in a non-circular way

a $\leftarrow$ **not** b
b $\leftarrow$ **not** c
d

{d, b}

fly(X) $\leftarrow$ bird(X), **not** ab(X)
ab(X) $\leftarrow$ penguin(X)
bird(X) $\leftarrow$ penguin(X)
bird(tweety)
penguin(skippy)

{penguin(skippy), bird(tweety),
bird(skippy), ab(skippy), fly(tweety)}

p $\leftarrow$ a
a $\leftarrow$ **not** b
b $\leftarrow$ **not** a

{p, a}, {b}

p $\leftarrow$ p
q $\leftarrow$ **not** p

{q}

Definition. [Lifschitz/Pearce/Valverde 2001]
Programs P and Q are ***strongly equivalent*** iff
for all programs X it holds that $P \cup X$ and $Q \cup X$ have the same answer sets

- Justifies **replacing a subset of rules** while preserving overall semantics

p ← **not** q	p

- Equivalent: both have the same single answer set {p}
- But not strongly equivalent: if we add q we get {q} and {p, q}, rsp.

p ← q q	p q

- These are strongly equivalent

p ← q, **not** q	

- Strongly equivalent to the empty program

Strong Equivalence can be Represented as Classical First-Order Equivalence

- For each **program predicate** p we have two **logic predicates** p^0, p^1
- Representing a logic with two worlds: *here* p^0 and *there* p^1
- Representing a three valued logic:

p is false	$\neg p^0 \wedge \neg p^1$
p is not false	$\neg p^0 \wedge p^1$
p is true	$p^0 \wedge p^1$

Definition (Sketch). For a rule

$$R \;=\; \mathsf{p}(X); \mathbf{not}\ \mathsf{q}(X) \leftarrow \mathsf{r}(X), \mathbf{not}\ \mathsf{s}(X)$$

define

$$\gamma^0(R) \stackrel{\text{def}}{=} \forall x\, (\mathsf{r}^0(x) \wedge \neg\mathsf{s}^1(x) \rightarrow \mathsf{p}^0(x) \vee \neg\mathsf{q}^1(x))$$
$$\gamma^1(R) \stackrel{\text{def}}{=} \forall x\, (\mathsf{r}^1(x) \wedge \neg\mathsf{s}^1(x) \rightarrow \mathsf{p}^1(x) \vee \neg\mathsf{q}^1(x))$$

For a program P define

$$\gamma(P) \stackrel{\text{def}}{=} \bigwedge_{R \in P} \gamma^0(R) \wedge \bigwedge_{R \in P} \gamma^1(R)$$

For a program P define

$$\mathsf{S}_P \stackrel{\text{def}}{=} \bigwedge_{p \in \mathcal{P}red(P)} \forall \mathbf{x} (p^0(\mathbf{x}) \rightarrow p^1(\mathbf{x}))$$

Proposition. [Lin 2002, Pearce/Tompits/Woltran 2009, Ferraris/Lee/Lifschitz 2011, Heuer 2020]

Programs P and Q are **strongly equivalent** iff

$$\mathsf{S}_{P \cup Q} \wedge \gamma(P) \;\equiv\; \mathsf{S}_{P \cup Q} \wedge \gamma(Q)$$

Making Precise Our Question for Synthesis of Logic Programs

$$P \models Q \leftrightarrow R$$
$$P \land Q \;\models\; R \;\models\; P' \rightarrow Q'$$

- Idea: For given logic programs P, Q and vocabulary V **synthesize** a program R **in** V that is **"equivalent"** to Q under assumptions P
- But: Logic programs are considered under **nonmonotonic semantics**

Task. For given programs P, Q and vocabulary V (a set of predicates) compute a program R **in** V s.th. $P \cup R$ is **strongly equivalent** to $P \cup Q$

- We consider strong equivalence wrt. a "background program" P, which may be empty
- R in V and for all programs X it holds that $X \cup P \cup Q$ and $X \cup P \cup R$ have the same answer sets

Outline of Our Approach to the Synthesis of Logic Programs

Task. For given programs P, Q and vocabulary V (a set of predicates) compute a program R **in** V s.th. $P \cup R$ is **strongly equivalent** to $P \cup Q$

1. Develop a first-order **characterization of first-order formulas that encode a logic program**

2. Develop a **method to decode such a formula** into a program, up to strong equivalence

3. Develop a variation of **Craig interpolation for formulas that encode logic programs**

4. On its basis, show a **projective Beth theorem for logic programs**
   - It inherits **effectivity** and **practical implementations** from Craig interpolation
   - Its effective version realizes the considered task

5. A refinement gives some control on allowed **positions in rule components** of predicates in R
   (head | body) × (positive | negated)

Definition. $\mathsf{rename}_{0\mapsto 1}(F)$ is F with 0-superscripted predicates p^0 replaced by the corresponding 1-superscripted predicates p^1

Definition. F ***encodes a program*** iff F is universal and $\mathsf{S}_F \land F \models \mathsf{rename}_{0\mapsto 1}(F)$

Theorem: Formulas Encoding a Logic Program.

(i) For all programs P: $\gamma(P)$ **encodes a program**

(ii) **If F encodes a program, then there is a program P** s.th.

(1) $\mathsf{S}_F \models \gamma(P) \leftrightarrow F$

(2) $\mathcal{P}red(P) \subseteq \mathcal{P}red^{LP}(F)$

(3) $\mathcal{F}un(P) \subseteq \mathcal{F}un(F)$

Moreover, such a program P can be **effectively constructed** from F

Proof. Procedure that extracts P from given F

On the Decoding Procedure

- For given F that encodes a program ($\mathsf{S}_F \land F \models \mathsf{rename}_{0\mapsto 1}(F)$), returns a program P s.th.

$$\mathsf{S}_F \models \gamma(P) \leftrightarrow F$$

- Converts the formula to CNF and **basically converts each clause to a program rule**
- Clauses that meet a **special criterion can be omitted** in the rule conversion
- Optional preprocessing where strong equivalence of the represented program is preserved

F	P	
$\neg p^0 \lor q^1 \lor r^0$	$r \leftarrow p, \textbf{not}\ q$	
$\neg p^1 \lor q^1 \lor r^1$	$\textbf{not}\ p \leftarrow \textbf{not}\ q, \textbf{not}\ r$	Rule can be omitted
$\neg s^1 \lor t^1 \lor u^1$	$\textbf{not}\ s \leftarrow \textbf{not}\ t, \textbf{not}\ u$	

F	Does not encode a logic program
$\neg p^0 \lor q^1 \lor r^0$	

Definition. A ***Craig-Lyndon interpolant*** of F and G s.th. $F \models G$ is a formula H s.th.

1. $F \models H$
2. $H \models G$
3. $\mathcal{Voc}(H) \subseteq \mathcal{Voc}(F) \cap \mathcal{Voc}(G)$, taking also **polarity** of predicate occurrences into account

Theorem: LP-Interpolation. Let F encode a logic program, and let G be s.th. $\mathcal{Fun}(F) \subseteq \mathcal{Fun}(G)$ and $\mathsf{S}_F \land F \models \mathsf{S}_G \rightarrow G$. Then there exists a first-order formula H, the ***LP-interpolant*** of F and G, s.th.

1. $\mathsf{S}_F \land F \models H$
2. $H \models \mathsf{S}_G \rightarrow G$
3. $\mathcal{Pred}^{\pm}(H) \subseteq S \cup \{+p^1 \mid +p^0 \in S\} \cup \{-p^1 \mid -p^0 \in S\}$, where $S = \mathcal{Pred}^{\pm}(\mathsf{S}_F \land F) \cap \mathcal{Pred}^{\pm}(\mathsf{S}_G \rightarrow G)$
4. $\mathcal{Fun}(H) \subseteq \mathcal{Fun}(F)$
5. H encodes a logic program

Moreover, such an H can be effectively constructed via Craig-Lyndon interpolation applied to $\mathsf{S}_F \land F$ and $\mathsf{S}_G \rightarrow G$

Proof. Let H' be a Craig-Lyndon interpolant of $\mathsf{S}_F \land F$ and $\mathsf{S}_G \rightarrow G$. Define $H \stackrel{\text{def}}{=} H' \land \text{rename}_{0\mapsto 1}(H')$

Theorem: Effective Projective Definability of Logic Programs. Let P and Q be programs and let $V \subseteq \mathcal{Pred}(P) \cup \mathcal{Pred}(Q)$ be a set of predicates. The **existence** of a program R s.th.

1. $\mathcal{Pred}(R) \subseteq V$
2. $\mathcal{Fun}(R) \subseteq \mathcal{Fun}(P) \cup \mathcal{Fun}(Q)$
3. $P \cup R$ and $P \cup Q$ are **strongly equivalent**

is **expressible as entailment between two first-order formulas**

Moreover, such a program R can be **effectively constructed** via Craig-Lyndon interpolation applied to both sides of the entailment

Proof. The entailment that characterizes existence of a logic program R is

$$\mathsf{S}_P \land \mathsf{S}_Q \land \gamma(P) \land \gamma(Q) \;\models\; \neg\mathsf{S}_{P'} \lor \neg\mathsf{S}_{Q'} \lor \neg\gamma(P') \lor \gamma(Q'),$$

where the primed P' and Q' are like P and Q, except that predicates not in V are replaced by fresh predicates

If the entailment holds, we can construct a program R as follows: Let H be the LP-interpolant of $\gamma(P) \land \gamma(Q)$ and $\neg\gamma(P') \lor \gamma(Q')$ and extract the program R from H with our procedure

Effective Projective Definability of Logic Programs – Basic Examples

For given P, Q, V, find a program R s.th.

1. $\mathcal{P}red(R) \subseteq V$
2. $\mathcal{F}un(R) \subseteq \mathcal{F}un(P) \cup \mathcal{F}un(Q)$
3. $P \cup R$ and $P \cup Q$ are strongly equivalent

$$Q = \begin{array}{l} \mathsf{p} \leftarrow \mathsf{q}, \mathsf{r} \\ \mathsf{p}; \mathsf{q} \leftarrow \mathsf{r} \\ \mathsf{q} \leftarrow \mathsf{q}, \mathsf{s} \end{array} \qquad V = \{\mathsf{p}, \mathsf{r}\}$$

$$R = \mathsf{p} \leftarrow \mathsf{r}$$

$$P = \mathsf{p}(X) \leftarrow \mathsf{q}(X) \qquad Q = \begin{array}{l} \mathsf{r}(X) \leftarrow \mathsf{p}(X) \\ \mathsf{r}(X) \leftarrow \mathsf{q}(X) \end{array} \qquad V = \{\mathsf{p}, \mathsf{r}\}$$

$$R = \mathsf{r}(X) \leftarrow \mathsf{p}(X)$$

$$P = \leftarrow \mathsf{p}(X), \mathsf{q}(X) \qquad Q = \mathsf{r}(X) \leftarrow \mathsf{p}(X), \mathbf{not}\ \mathsf{q}(X) \qquad V = \{\mathsf{p}, \mathsf{r}\}$$

$$R = \mathsf{r}(X) \leftarrow \mathsf{p}(X)$$

For given P, Q, V, find a program R s.th.

1. $\mathcal{P}red(R) \subseteq V$
2. $\mathcal{F}un(R) \subseteq \mathcal{F}un(P) \cup \mathcal{F}un(Q)$
3. $P \cup R$ and $P \cup Q$ are strongly equivalent

$$P = \begin{array}{l} \mathsf{p}(X) \leftarrow \mathsf{q}(X), \mathbf{not}\ \mathsf{r}(X) \\ \mathsf{p}(X) \leftarrow \mathsf{s}(X) \\ \mathbf{not}\ \mathsf{r}(X); \mathsf{s}(X) \leftarrow \mathsf{p}(X) \\ \mathsf{q}(X); \mathsf{s}(X) \leftarrow \mathsf{p}(X) \end{array}$$

$$Q = \mathsf{t}(X) \leftarrow \mathsf{p}(X)$$

$$R = \begin{array}{l} \mathsf{t}(X) \leftarrow \mathsf{q}(X), \mathbf{not}\ \mathsf{r}(X) \\ \mathsf{t}(X) \leftarrow \mathsf{s}(X) \end{array}$$

$$V = \{\mathsf{q}, \mathsf{r}, \mathsf{s}, \mathsf{t}\}$$

- Idea: P expresses a schema mapping from client predicate p to KB predicates q, r, s
  The result R is a rewriting of the client query Q in terms of KB predicates
- Only the first two rules of P actually describe the mapping, the other two complete them
- Effects unfolding of p
- Also works with R and Q switched and $V = \{\mathsf{p}, \mathsf{t}\}$: then it effects folding into p

Corollary: Position-Constrained Effective Projective Definability of Logic Programs. Our definability theorem holds in a strengthened variation where three sets V_+, V_{+1}, V_- of predicates are given to the effect that a predicate p can occur in the respective component of a rule of R only if it is a member of a set of predicates according to the following table

p is allowed in	only if p is in
Positive heads	V_+
Negative bodies	$V_+ \cup V_{+1}$
Negative heads	V_-
Positive bodies	V_-

$$P = \mathsf{p} \leftarrow \mathsf{q} \qquad Q = \begin{array}{l} \mathsf{r} \leftarrow \mathsf{p} \\ \mathsf{r} \leftarrow \mathsf{q} \\ \mathsf{q} \leftarrow \mathsf{s} \end{array} \qquad \begin{array}{l} V_+ = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\} \\ V_{+1} = \{\} \\ V_- = \{\mathsf{p}, \mathsf{r}, \mathsf{s}\} \end{array}$$

$$R = \begin{array}{l} \mathsf{r} \leftarrow \mathsf{p} \\ \mathsf{q} \leftarrow \mathsf{s} \end{array}$$

$$P = \mathsf{p} \leftarrow \mathsf{q} \qquad Q = \begin{array}{l} \leftarrow \mathsf{q}, \mathbf{not}\ \mathsf{p} \\ \mathsf{r} \leftarrow \mathsf{q} \\ \mathsf{s} \leftarrow \mathsf{p} \end{array} \qquad \begin{array}{l} V_+ = \{\mathsf{q}, \mathsf{r}, \mathsf{s}\} \\ V_{+1} = \{\} \\ V_- = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\} \end{array}$$

$$R = \begin{array}{l} \mathsf{r} \leftarrow \mathsf{q} \\ \mathsf{s} \leftarrow \mathsf{p} \end{array}$$

$$P = \begin{array}{l} \mathsf{p} \leftarrow \mathsf{q} \\ \mathsf{r} \leftarrow \mathsf{p} \end{array} \qquad Q = \begin{array}{l} \mathsf{s} \leftarrow \mathbf{not}\ \mathsf{r} \\ \mathsf{r} \leftarrow \mathsf{q} \end{array} \qquad \begin{array}{l} V_+ = \{\mathsf{s}\} \\ V_{+1} = \{\mathsf{r}\} \\ V_- = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\} \end{array}$$

$$R = \mathsf{s} \leftarrow \mathbf{not}\ \mathsf{r}$$

Prototypical Implementation

- Implemented in ***PIE (Proving, Interpolating, Eliminating)*** [W 2016], embedded in **SWI-Prolog**
- Craig-Lyndon interpolation is done with first-order provers

CMProver
[W 1992–] similar to *PTTP*, *SETHEO*, *leanCoP*

Prover9
+ Prooftrans

Clausal tableau

Binary resolution proof

Clausal tableau in cut normal form
(semantic tree)

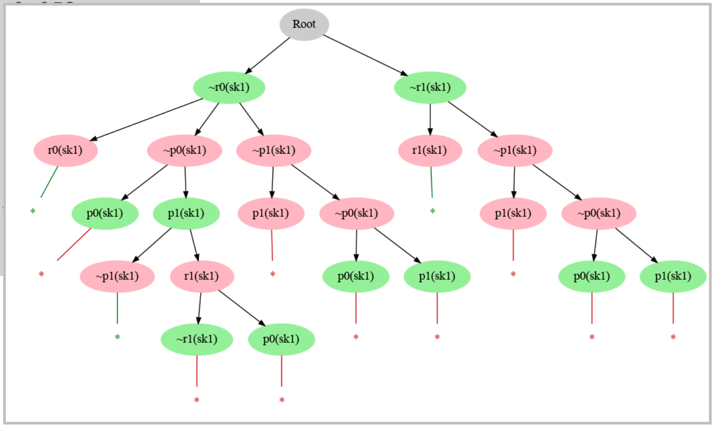
Clausal tableau in hyper form [W 2023]
(leaves = neg. literals)

Craig-Lyndon interpolation for clausal tableaux [W 2021]

- *Vampire* and *E* do not emit gap-free resolution proofs suited for interpolation,
  but proof tasks underlying interpolation can be tried with any prover supporting TPTP FOF
- Simplifications are important at all stages
- Nice Skolemization is useful: $\forall \overline{y}\, P(\overline{y}) \land \forall \overline{y}\, Q(\overline{y}) \land \forall \overline{y}\, P'(\overline{y}) \land \exists \overline{x}\, \neg Q'(\overline{x})$
  Not by default CNF trafos of *PIE*, *Prover9*, *E*, *Vampire* (but no problem for *Vampire*)

Prototypical Implementation – Hands-On

```
?- exdef(14-3, P, Q, V), p_def(P, Q, V, R, []).
% depth           0          0.122 msec
% depth           1          0.074 msec
% depth           2          0.050 msec
% depth           3          0.062 msec
% depth           4
% depth           5
% depth           6
% depth           7
% depth           8
% ----- solution after
P = [(false<--p(_A), q(_A))],
Q = [(r(_A)<--p(_A), not q(_A))],
V = [p, r],
R = [(r(_B)<--p(_B))]
```



Agenda (I): Relate to Direct Interpolation for Non-Classical Logics

- Related works: [Applications of] Craig interpolation and Beth definability for **equilibrium logic**, based on earlier (mostly existential) results on interpolation in non-classical logics [Gabbay/Pearce/Valverde 2011, Pearce/Valverde 2012]
- The logic underlying strong equivalence is HT, aka Gödel's G_3 – does it have **feasible** interpolation?
- Our LP-interpolation theorem can be rephrased in terms of **interpolation for logic programs** (see current version of implementation)
- Can our approach be transferred to obtain a feasible interpolation method for HT/G_3?
- Known: Uniform interpolation for G_3 [Baaz/Veith 1999]
- In principle related, but apparently so far completely Beth-unaware: forgetting in ASP

Agenda (II): Generalizations and Refinements

- **Safety** (roughly: all variables of a rule have an occurence in the positive body)
  – related to range-restriction [W 2023]
- Disallowing constants or **function** symbols
  – but Craig interpolation introduces existential quantifiers for “left-only” such symbols
- **Arithmetics**, **theories**, **aggregation**
  – current topics in verification of strong equivalence
- **Restrictions on rule form** (e.g. no negative head, a single positive head)
  – related to Horn [W 2023]
- Transfer to **completion-based program encodings**
- **Hidden predicates** (which may have an arbitrary extension in R)
  – relative equivalence [Lin 2002], projected answer sets [Eiter et al. 2005], external behavior [Fandinno et al. 2023]
- **“Schema mappings”** with the involved completion
  – possibly related to [Toman/Wedell 2023]
- Applying our encoding/decoding to **program simplification** via first-order formula simplification

Conclusion – Generalizing Summary

Task. For given programs P, Q and vocabulary V (a set of predicates) compute a program R **in** V s.th. $P \cup R$ is **strongly equivalent** to $P \cup Q$

$$P \models Q \leftrightarrow R$$
$$P \land Q \;\models\; R \;\models\; P' \rightarrow Q'$$

- **Equivalence notion in the target logic** (strong equivalence), **expressed as classical equivalence**
  - Target expressions are **encoded** as classical representation (of a logic with two worlds, p^0 and p^1 for each p)
  - The classical equivalence is modulo certain axioms ($p^0 \rightarrow p^1$)
- Encoded target expressions can be **decoded**, modulo the equivalence notion, without enriching the vocabulary
- **Classical Craig interpolation on encoded target expressions plus postprocessing yields an encoded target expression**
- Together with the decoding we obtain a **projective Beth property for the target logic**
- I.e. we can **synthesize target expressions** R from given target expressions P, Q and vocabulary V
- **Effectivity, feasibility, also practical, is inherited from Craig interpolation for classical logic**

References I

[Baaz and Veith, 1999] Baaz, M. and Veith, H. (1999).
Interpolation in fuzzy logic.
Ann. Math. Logic, 38:461–489.

[Baral, 2010] Baral, C. (2010).
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press.

[Cabalar and Ferraris, 2007] Cabalar, P. and Ferraris, P. (2007).
Propositional theories are strongly equivalent to logic programs.
Theory Pract. Log. Program., 7(6):745–759.

[Delgrande, 2017] Delgrande, J. P. (2017).
A knowledge level account of forgetting.
JAIR, 60:1165–1213.

References II

[Eiter et al., 2005] Eiter, T., Tompits, H., and Woltran, S. (2005).
On solution correspondences in answer-set programming.
In Kaelbling, L. P. and Saffiotti, A., editors, *IJCAI-05*, pages 97–102. Professional Book Center.

[Fandinno et al., 2023] Fandinno, J., Hansen, Z., Lierler, Y., Lifschitz, V., and Temple, N. (2023).
External behavior of a logic program and verification of refactoring.
Theory Pract. Log. Program., 23(4):933–947.

[Fandinno and Lifschitz, 2023] Fandinno, J. and Lifschitz, V. (2023).
On Heuer's procedure for verifying strong equivalence.
In Gaggl, S. A., Martinez, M. V., and Ortiz, M., editors, *JELIA 2023*, volume 14281 of *LNCS*, pages 253–261. Springer.

[Ferraris et al., 2011] Ferraris, P., Lee, J., and Lifschitz, V. (2011).
Stable models and circumscription.
Artif. Intell., 175(1):236–263.

References III

[Gabbay et al., 2011] Gabbay, D. M., Pearce, D., and Valverde, A. (2011).
Interpolable formulas in equilibrium logic and answer set programming.
JAIR, 42:917–943.

[Gelfond and Lifschitz, 1988] Gelfond, M. and Lifschitz, V. (1988).
The stable model semantics for logic programming.
In Kowalski, R. A. and Bowen, K. A., editors, *ICLP/SLP*, pages 1070–1080, Cambridge, MA. MIT Press.

[Gonçalves et al., 2023] Gonçalves, R., Knorr, M., and Leite, J. (2023).
Forgetting in answer set programming - A survey.
Theory Pract. Log. Program., 23(1):111–156.

[Heuer, 2020] Heuer, J. (2020).
Automated verification of equivalence properties in advanced logic programs.
Bachelor's thesis, University of Potsdam.

[Heuer, 2023] Heuer, J. (2023).
Automated verification of equivalence properties in advanced logic programs.
In Schwarz, S. and Wenzel, M., editors, *WLP 2023*.

References IV

[Heuer and Wernhard, 2024] Heuer, J. and Wernhard, C. (2024).

Synthesizing strongly equivalent logic programs: Beth definability for answer set programs via Craig interpolation in first-order logic.

In Benzmüller, C., Heule, M., and Schmidt, R., editors, *IJCAR 2024*, LNCS (LNAI). Springer.

To appear, preprint: `https://arxiv.org/abs/2402.07696`.

[Lifschitz, 2010] Lifschitz, V. (2010).

Thirteen definitions of a stable model.

In Blass, A., Dershowitz, N., and Reisig, W., editors, *Fields of Logic and Computation, Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *LNCS*, pages 488–503. Springer.

[Lifschitz, 2019] Lifschitz, V. (2019).

Answer Set Programming.

Springer.

[Lifschitz et al., 2001] Lifschitz, V., Pearce, D., and Valverde, A. (2001).

Strongly equivalent logic programs.

ACM Trans. Comp. Log., 2(4):526–541.

[Lin, 2002] Lin, F. (2002).
Reducing strong equivalence of logic programs to entailment in classical propositional logic.
In *KR-02*, pages 170–176. Morgan Kaufmann.

[McCune, 2010] McCune, W. (2005–2010).
Prover9 and Mace4.
`http://www.cs.unm.edu/~mccune/prover9`, accessed Feb 5, 2024.

[Pearce et al., 2009] Pearce, D., Tompits, H., and Woltran, S. (2009).
Characterising equilibrium logic and nested logic programs: Reductions and complexity.
Theory Pract. Log. Program., 9(5):565–616.

[Pearce and Valverde, 2012] Pearce, D. and Valverde, A. (2012).
Synonymous theories and knowledge representations in answer set programming.
J. Comput. Syst. Sci., 78(1):86–104.

[Toman and Weddell, 2022] Toman, D. and Weddell, G. E. (2022).
First order rewritability in ontology-mediated querying in horn description logics.
In *AAAI 2022, IAAI 2022, EAAI 2022*, pages 5897–5905. AAAI Press.

References VI

[Wernhard, 2016] Wernhard, C. (2016).

The PIE system for proving, interpolating and eliminating.

In Fontaine, P., Schulz, S., and Urban, J., editors, *PAAR 2016*, volume 1635 of *CEUR Workshop Proc.*, pages 125–138. CEUR-WS.org.

[Wernhard, 2021] Wernhard, C. (2021).

Craig interpolation with clausal first-order tableaux.

J. Autom. Reasoning, 65(5):647–690.

[Wernhard, 2023] Wernhard, C. (2023).

Range-restricted and Horn interpolation through clausal tableaux.

In Ramanayake, R. and Urban, J., editors, *TABLEAUX 2023*, volume 14278 of *LNCS (LNAI)*, pages 3–23. Springer.

On the Decoding Procedure

- For given F that encodes a program ($\mathsf{S}_F \land F \models \text{rename}_{0\mapsto 1}(F)$), returns a program P s.th.

$$\mathsf{S}_F \models \gamma(P) \leftrightarrow F$$

- Converts the formula to CNF and **basically converts each clause to a program rule**
- Clauses that meet a **special criterion can be omitted** in the rule conversion
- Optional preprocessing where strong equivalence of the represented program is preserved

F	P	
$\neg p^0 \lor q^1 \lor r^0$	$r \leftarrow p, \mathbf{not}\ q$	
$\neg p^1 \lor q^1 \lor r^1$	$\mathbf{not}\ p \leftarrow \mathbf{not}\ q, \mathbf{not}\ r$	Rule can be omitted
$\neg s^1 \lor t^1 \lor u^1$	$\mathbf{not}\ s \leftarrow \mathbf{not}\ t, \mathbf{not}\ u$	

F	Does not encode a logic program
$\neg p^0 \lor q^1 \lor r^0$	

Procedure: Extracting a Program from a Formula.

1. **Bring the input F into a CNF $\forall \mathbf{x}\,(M_0 \wedge M_1)$** s.th.
   - all clauses of M_0 have a 0-literal and
   - all clauses of M_1 have only 1-literals
2. **Partition M_1 into M_1', M_1'' s.th.**
   $\forall \mathbf{x}\ \mathsf{rename}_{0\mapsto 1}(M_0) \models \forall \mathbf{x}\ M_1''$
   E.g. take $M_1' = M_1$ and $M_1'' = \top$
   Or place each clause C in M_1 into M_1'' or M_1' depending on whether there is a D in M_0 s.th. $\mathsf{rename}_{0\mapsto 1}(D)$ subsumes C
3. **Return as P the set of rules**
   $A; \mathbf{not}\ B \leftarrow C, \mathbf{not}\ D$ for each clause
   $C^0 \wedge \neg D^1 \rightarrow A^0 \vee \neg B^1$ in $M_0 \wedge M_1'$

Option: Preprocess the input F to F' s.th.
$\mathcal{V}oc(F') \subseteq \mathcal{V}oc(F)$ **and** $\mathsf{S}_F \models F' \leftrightarrow F$

F	P
$\neg \mathsf{p}^0 \vee \mathsf{q}^1 \vee \mathsf{r}^0$	$\mathsf{r} \leftarrow \mathsf{p}, \mathbf{not}\ \mathsf{q}$
$\neg \mathsf{p}^1 \vee \mathsf{q}^1 \vee \mathsf{r}^1$	$\mathbf{not}\ \mathsf{p} \leftarrow \mathbf{not}\ \mathsf{q}, \mathbf{not}\ \mathsf{r}$
$\neg \mathsf{s}^1 \vee \mathsf{t}^1 \vee \mathsf{u}^1$	$\mathbf{not}\ \mathsf{s} \leftarrow \mathbf{not}\ \mathsf{t}, \mathbf{not}\ \mathsf{u}$

$C_1 = \neg \mathsf{p}^0 \vee \mathsf{q}^1 \vee \mathsf{r}^0 \quad R_1 = \mathsf{r} \leftarrow \mathsf{p}, \mathbf{not}\ \mathsf{q}$
$C_2 = \neg \mathsf{p}^0 \vee \mathsf{q}^1 \vee \mathsf{r}^1 \quad R_2 = \leftarrow \mathsf{p}, \mathbf{not}\ \mathsf{q}, \mathbf{not}\ \mathsf{r}$
$C_3 = \neg \mathsf{p}^1 \vee \mathsf{q}^1 \vee \mathsf{r}^1$

By **preprocessing** F we can eliminate C_2
The rule for C_3 can be omitted.