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

CIBD Workshop

Amsterdam, April 23, 2024

- 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 the **answer sets** (stable models [Gelfond/Lifschitz 1988]) of a given program
- These are minimal Herbrand models in which all facts are properly justified in a non-circular way

a ← **not** b
b ← **not** c
d

{d, b}

fly(X) ← bird(X), **not** ab(X)
ab(X) ← penguin(X)
bird(X) ← penguin(X)
bird(tweety)
penguin(skippy)

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

p ← a
a ← **not** b
b ← **not** a

{p, a}, {b}

p ← p
q ← **not** p

{q}

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

- Justifies replacability of a subset of the rules of a program such that its overall semantics, the set of answer sets or stable models, is preserved

$\mathsf{p} \leftarrow \mathbf{not}\ \mathsf{q}$	p

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

$\mathsf{p} \leftarrow \mathsf{q}$ q	p q

- These are strongly equivalent

$\mathsf{p} \leftarrow \mathsf{q}, \mathbf{not}\ \mathsf{q}$	

- 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, reflecting a modal logic with two states

Definition (here by example). For a rule

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

define

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

For a program P define

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

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

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

Our Objective: Craig and Projective Beth for Logic Programs

Task. For given programs P, Q and vocabulary V (a set of predicates)
find 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 S it holds that $S \cup P \cup Q$ and $S \cup P \cup R$ have the same answer sets

Available Tools

- The γ **encoding of programs to express strong equivalence as a first-order equivalence**
- **Construction of a first-order definition by Craig interpolation**, also practically by first-order ATP systems

Our Approach

1. Develop a first-order **criterion to check whether a formula encodes a logic program**
2. Develop a **method to decode a formula that encodes a program** into a program, up to strong equivalence
3. Develop a variation of **Craig-Lyndon interpolation for formulas that encode logic programs**
4. On its basis, show a **projective Beth theorem for logic programs**
   - Its inherits **effectivity** from Craig-Lyndon interpolation (also practical implementations)
   - Its effective version realizes the above task
5. A refinement gives some control on allowed **rule components** (head, body, positive, negated) of predicates in R

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

$\mathrm{rename}_{0\mapsto 1}$ preserves entailment and thus also equivalence:
If $F \models G$, then $\mathrm{rename}_{0\mapsto 1}(F) \models \mathrm{rename}_{0\mapsto 1}(G)$
If $F \equiv G$, then $\mathrm{rename}_{0\mapsto 1}(F) \equiv \mathrm{rename}_{0\mapsto 1}(G)$

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

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

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 from a proof of $\mathsf{S}_F \land F \models \mathsf{S}_G \rightarrow G$

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 $V \subseteq \mathcal{P}red(P) \cup \mathcal{P}red(Q)$ be a set of predicates. The **existence** of 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**

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

Moreover, if for given P, Q, V a program R with these properties exists, such a program can be **effectively constructed** from a proof 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

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

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

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

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

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

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

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

Effective Projective Definability of Logic Programs – "Schema Mapping" Examples

For given P, Q, V, find 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

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

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

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

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

- Idea: P expresses a schema mapping from client predicate p to knowledge base predicates q, r, s
  The result R is a rewriting of the client query Q in terms of knowledge base predicates
- Only the first two rules of P actually describe the mapping, the other two complete them

$P =$ As above

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

$$V = \{\mathsf{p}, \mathsf{t}\}$$

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

- While the first example realizes unfolding of p, the second realizes folding into p

Constraining Positions of Predicates within Rules

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_-

$P = \mathsf{p} \leftarrow \mathsf{q}$

$Q = \mathsf{r} \leftarrow \mathsf{p}$
$\quad\ \ \mathsf{r} \leftarrow \mathsf{q}$
$\quad\ \ \mathsf{q} \leftarrow \mathsf{s}$

$R = \mathsf{r} \leftarrow \mathsf{p}$
$\quad\ \ \mathsf{q} \leftarrow \mathsf{s}$

$V_+ = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\}$
$V_{+1} = \{\}$
$V_- = \{\mathsf{p}, \mathsf{r}, \mathsf{s}\}$

$P = \mathsf{p} \leftarrow \mathsf{q}$

$Q = \leftarrow \mathsf{q}, \mathbf{not}\ \mathsf{p}$
$\quad\ \ \mathsf{r} \leftarrow \mathsf{q}$
$\quad\ \ \mathsf{s} \leftarrow \mathsf{p}$

$R = \mathsf{r} \leftarrow \mathsf{q}$
$\quad\ \ \mathsf{s} \leftarrow \mathsf{p}$

$V_+ = \{\mathsf{q}, \mathsf{r}, \mathsf{s}\}$
$V_{+1} = \{\}$
$V_- = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\}$

$P = \mathsf{p} \leftarrow \mathsf{q}$
$\quad\ \ \mathsf{r} \leftarrow \mathsf{p}$

$Q = \mathsf{s} \leftarrow \mathbf{not}\ \mathsf{r}$
$\quad\ \ \mathsf{r} \leftarrow \mathsf{q}$

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

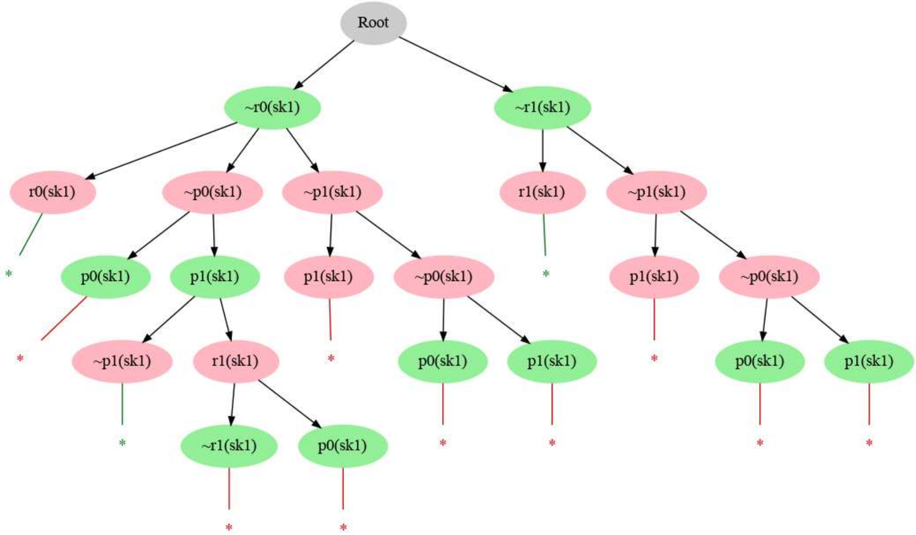
$V_+ = \{\mathsf{s}\}$
$V_{+1} = \{\mathsf{r}\}$
$V_- = \{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\}$

- Implemented in the ***PIE (Proving, Interpolating, Eliminating)*** environment [W 2016], embedded in **SWI-Prolog**
- Options for Craig interpolation, may lead to different solutions
  - ***CMProver*** (clausal tableaux/connection method, included in *PIE*)
    + interpolation for clausal tableaux [W 2021]
  - ***CMProver***
    + proof translation to preserve range restriction [W 2023]
    + interpolation for clausal tableaux [W 2021]
  - ***Prover9***
    + resolution proof translation [W 2023]
    + interpolation for clausal tableaux [W 2021]
- Vocabularies may also be specified complementary, like "forgetting"

```
?- 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       0.053 msec
% depth          5       0.062 msec
% depth          6       0.068 msec
% depth          7       0.140 msec
% depth          8
% ----- solution after   0.102 msec.
P = [(false<--p(_A), q(_A))],
Q = [(r(_A)<--p(_A), not q(_A))],
V = [p, r],
R = [(r(_B)<--p(_B))]
```

Prototypical Implementation – Screenshot with Proof

```
?- exdef(14-3, P, Q, V), p_def(P, Q, V, R, [ip_dotgraph='/tmp/proof.png',lpip_simp_input=2]).
```



Related Work, Some Open Issues, Desiderata

Related Work

- Craig interpolation and Beth for equilibrium logic with **existential** results [Gabbay/Pearce/Valverde 2011, Pearce/Valverde 2012]
- Maybe related: works on forgetting in ASP

Potential Generalizations and Refinements

- Disallowing constants or **function** symbols
  – but Craig interpolation introduces existential quantifiers for "left-only" such symbols
- **Safety** (roughly: all variables of a rule have an occurence in the positive body)
  – related to range-restriction [W 2023]
- **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]

More

- Applying the first-order coding/decoding to **program simplification**
- Is the general approach applicable elsewhere, e.g., robustness under replacement?

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

- An **equivalence notion in the target logic** (strong equivalence), **expressible as classical first-order equivalence**
  - Target expressions are **encoded** as classical representation of a logic with two states (p^0, p^1 for each p)
  - The classical equivalence is modulo specific axioms ($p^0 \rightarrow p^1$)
- Encoded target expressions can be **decoded**, modulo the equivalence notion, without enriching the vocabulary
- **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, even practical, is inherited from Craig interpolation**

Craig Interpolation and Beth Definability in a Nutshell

Definition. Formula Qx is ***implicitly definable*** in terms of vocabulary V within sentence K iff

$$K \land K' \vDash \forall x\,(Qx \leftrightarrow Q'x), \qquad \textit{(ImpDef)}$$

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

- *(ImpDef)* says that if two models of K agree on values of symbols in V, then they agree on the extension of Q

Definition. Formula Qx is ***explicitly definable*** in terms of vocabulary V within sentence K iff
there exists a formula Rx in the vocabulary V s.th. $K \vDash \forall x\,(Qx \leftrightarrow Rx)$ *(ExpDef)*

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

1. $F \vDash H$
2. $H \vDash G$
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 $F \vDash G$
[Beth 1953] In first-order logic (*ImpDef*) and (*ExpDef*) are equivalent

Proof of [Beth]. Write (*ImpDef*) as $K \land Qx \vDash K' \rightarrow Q'x$
Obtain Rx as Craig interpolant of $K \land Qx$ and $K' \rightarrow Q'x$

$$K \vDash \forall x\,(Qx \leftrightarrow Rx)$$

$$K \vDash Qx \rightarrow Rx \qquad K \vDash Rx \rightarrow Qx$$

$$K \land Qx \vDash Rx \vDash K' \rightarrow Q'x$$

References I

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

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

[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.

References II

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

[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*.

[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.

References IV

[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.

References V

[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.

[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.