Canonical Logic Programs are Succinctly Incomparable with Propositional Formulas

Yuping Shen and **Xishun Zhao***
Institute of Logic and Cognition
Department of Philosophy
Sun Yat-sen University
510275 Guangzhou, P.R. China
{shyping, hsszxs}@mail.sysu.edu.cn

Abstract

Canonical (logic) programs (CP) refer to the class of *normal* programs (LP) augmented with connective *not not*, and are equally expressive as *propositional formulas* (PF). In this paper we address the question of whether CP and PF are *succinctly incomparable*. Our main result shows that the PARITY problem *only* has exponential CP representations, while it can be polynomially represented in PF. In other words, PARITY *separates* PF from CP. Simply speaking, this means that exponential size blowup is generally inevitable when translating a set of PF formulas into a (logically) equivalent CP program (without introducing new variables). Furthermore, since it has been shown by Lifschitz and Razborov that there is also a problem which separates CP from PF (assuming $\mathsf{P} \nsubseteq \mathsf{NC}^1/\mathsf{poly}$), it follows that the two formalisms are indeed succinctly incomparable.

1 Introduction

The relationship between *(logic) programs* under *answer set semantics* (ASP) (Lifschitz 2008; Brewka, Eiter, and Truszczynski 2011) and *propositional satisfiability* (SAT) (Biere et al. 2009) gains a lot of attention in the literature. In 2006, Lifschitz and Razborov proved that an exponential size blowup is generally inevitable when translating a *normal program* (LP) to a (logically) equivalent set of *propositional formulas* (PF) (without introducing additional variables). More precisely, they showed that (a variant of) the P-complete problem PathSystem (PATH) has polynomial size LP representations, however, it *cannot* be polynomially represented in PF (assuming $\mathsf{P} \nsubseteq \mathsf{NC}^1/\mathsf{poly}$) (Lifschitz and Razborov 2006), i.e., PATH *separates* LP from PF.

As noted in (Lifschitz and Razborov 2006), PF can be considered as a special case of the class of *(nondisjunctive) nested programs* (NLP, without classical negation $\neg$) (Lifschitz 1999). Therefore, NLP is *stronger* than PF in terms of the *succinctness* criterion (or the *"comparative linguistics" approach*) (Gogic et al. 1995):

> That is, we consider formalism A to be stronger than formalism B if and only if any knowledge base (KB) in B has an equivalent KB in A that is only polynomially longer, while there is a KB in A that can be translated to B only with an exponential blowup.

So the following footnote in (Lifschitz 2008) seems convincing:

> ...ASP appears to be stronger than SAT in the sense of the "comparative linguistics" approach to knowledge representation...

However, ASP involves many kinds of programs, the above statement probably needs further clarification. Particularly, the class of so-called *(nondisjunctive) canonical programs* (CP[1]) (Lifschitz 1999; Lee 2005; Lee, Lifschitz, and Yang 2013), is a minimal form of ASP that is equally expressive as PF, but looks more likely *not* succinctly stronger. So a question naturally arises: *Does there exist a problem that separates PF from CP?* If there is such a problem, then CP and PF are *succinctly incomparable* (assuming $\mathsf{P} \nsubseteq \mathsf{NC}^1/\mathsf{poly}$).

In this paper we address the question and give a *positive* answer. Our main result shows that the PARITY problem separates PF from CP. Simply speaking, this means exponential size blowup is generally inevitable when translating a set of PF formulas into an equivalent CP program. The PARITY problem asks whether a binary string contains an odd number of 1's, and it is well-known that (i) PARITY$\in$ $\mathsf{NC}^1/\mathsf{poly}$[2], i.e., it has polynomial PF representations; (ii) PARITY$\notin$ AC^0, i.e., it cannot be represented by *polynomial* size (boolean) circuits with *constant depth* and *unbounded fan-in* (Arora and Barak 2009; Jukna 2012).

To show PARITY separates PF from CP, we provide a procedure that simplifies every PARITY program Π into a shorter program Π' s.t. Π' is equivalent to its *completion* $Comp(\Pi')$ (Erdem and Lifschitz 2003). Such completions are essentially constant depth, unbounded fan-in circuits. According to PARITY$\notin$ AC^0, these circuits must be of exponential size, consequently, there are no polynomial size PARITY programs in CP. In the rest of the paper, we shall introduce the basic concepts, key steps of our proof and discuss the importance of succinctness research in the theory and practice of Knowledge Representation (KR).

*Corresponding author.

[1] Extends LP with connective *not not*.

[2] $\mathsf{NC}^1/\mathsf{poly}$ (or *non-uniform* NC^1) *exactly* contains languages computable (representable) by polynomial size PF formulas.

2 Background

Canonical Programs

The following notations are adopted from (Lifschitz 1999; Lee 2005). A *rule element* e is defined as

$$e := \top \mid \bot \mid x \mid not\ x \mid not\ not\ x$$

in which $\top, \bot$ are 0-ary connectives, x is a *(boolean) variable* (or an *atom*) and not is a unary connective[3]. A *(nondisjunctive canonical) rule* is an expression of the form

$$H \leftarrow B \tag{1}$$

where the *head* H is either a variable or the connective $\bot$, and the *body* B is a finite set of rule elements. A *canonical program* Π is a finite set of rules. E.g., the following is a canonical program:

$$\begin{array}{ll} x_1 \leftarrow not\ not\ x_1, & x_3 \leftarrow not\ x_1, not\ x_2, \\ x_2 \leftarrow not\ not\ x_2, & x_3 \leftarrow x_1,\ x_2. \end{array} \tag{2}$$

A canonical program Π is *normal* if it contains no connectives $not\ not$. A normal program Π is *basic* if it contains no occurrences of connective not.

The *satisfaction relation* $\models$ between a set of variables I and a rule element is defined as follows:

- $I \models \top$ and $I \not\models \bot$,
- $I \models x$ iff $I \models not\ not\ x$ iff $x \in I$,
- $I \models not\ x$ iff $x \notin I$.

Say I satisfies a set of rule elements B if I satisfies each rule element in B. We say I is *closed* under a program Π, if I is closed under every rule in Π, i.e., for each rule $H \leftarrow B \in \Pi$, $I \models H$ whenever $I \models B$. Let Π be a basic program and let $Cn(\Pi)$ denotes the *minimal* set (in terms of set inclusion) closed under Π, we say I is an *answer set* of Π if $I = Cn(\Pi)$. Note that a basic program has exactly one answer set.

The *reduct* Π^I of a program Π w.r.t. I is a set of rules obtained from Π via: (i) Replacing each $not\ not\ x$ with $\top$ if $I \models x$, and with $\bot$ otherwise; (ii) Replacing each $not\ x$ with $\top$ if $I \not\models x$, and with $\bot$ otherwise. Observe that Π^I must be a basic program. We say I is an answer set of Π if $I = Cn(\Pi^I)$, i.e., I is an answer set of Π^I. E.g., the following single rule program:

$$x \leftarrow not\ not\ x \tag{3}$$

has two answer sets $\emptyset$ and $\{x\}$.

For a set of rule elements B, define $var(B) = \{e \in B : e \text{ is a variable}\}$. E.g., $var(\{x_1, not\ x_2, not\ not\ x_3\}) = \{x_1\}$. The *signature* $sig(\Pi)$ of a program Π is the set of all involved variables in Π. By $Ans(\Pi)$ we denote the set of all answer sets of Π. E.g., if Π is (2), then $sig(\Pi) = \{x_1, x_2, x_3\}$ and $Ans(\Pi) = \{\{x_1, x_2, x_3\}, \{x_1\}, \{x_2\}, \{x_3\}\}$. The *size* $|\Pi|$ of Π is the number of its rules. As a convention, Π_n refers to a program with signature $\{x_1, \ldots, x_n\}$, i.e., $sig(\Pi_n) = \{x_1, \ldots, x_n\}$.

It is easy to see that by using rules of the form (3) and appropriate *constraints* of the form $\bot \leftarrow B$, it is easy to give an arbitrary set of answer sets over $sig(\Pi_n)$, in other words, CP has exactly the same expressive power as PF.

[3] By (Lifschitz 1999), $not\ not\ not\ x$ can be replaced by $not\ x$.

Problem Representation and Succinctness

A *(binary) string* is a finite sequence of *bits* from $\{0, 1\}$. A string w of length n (i.e., $w \in \{0,1\}^n$) defines a subset of variables $\{x_1, \ldots, x_n\}$. E.g., 1010 stands for $\{x_1, x_3\}$. Therefore, a set of variables I and a string w can be regarded as the same. A *problem* (or *language*) L is a set of strings.

Definition 2.1 (Problem Representation)**.** *A problem L can be* represented *in a class of programs (or formulas, etc) $\mathcal{C}$ (i.e., $L \in \mathcal{C}$), if there exists a* sequence *of programs $\{\Pi_n\}$ ($n = 1, 2, \ldots$) in $\mathcal{C}$ that* computes *L, i.e., for every string $w \in \{0,1\}^n$,*

$$w \in L \Leftrightarrow w \in Ans(\Pi_n).$$

Say L has polynomial representations in $\mathcal{C}$ (i.e., $L \in$ Poly-$\mathcal{C}$), if $L \in \mathcal{C}$ and $|\Pi_n|$ is bounded by a polynomial $p(n)$.

Recall that the PARITY problem is the set of binary strings with an odd number of 1's. By PARITY$_n$ we denote PARITY strings of length n. Clearly, (2) represents PARITY$_3$ since its answer sets are $111, 100, 010$ and 001.

Definition 2.2 (Succinctness (Gogic et al. 1995; French et al. 2013))**.** *Let $\mathcal{C}, \mathcal{C}'$ be two classes of programs such that for every problem L, $L \in \mathcal{C} \Leftrightarrow L \in \mathcal{C}'$. Say $\mathcal{C}$ is* at least as succinct as *$\mathcal{C}'$ (i.e., $\mathcal{C}' \preceq \mathcal{C}$), if for every problem L,*

$$L \in \textit{Poly-}\mathcal{C}' \Rightarrow L \in \textit{Poly-}\mathcal{C}.$$

If $L \in$ Poly-$\mathcal{C}$ but $L \notin$ Poly-$\mathcal{C}'$ (i.e., $\mathcal{C} \npreceq \mathcal{C}'$), then L separates *$\mathcal{C}$ from $\mathcal{C}'$. If $\mathcal{C}' \preceq \mathcal{C}$ and $\mathcal{C} \npreceq \mathcal{C}'$, then $\mathcal{C}$ is* strictly more succinct than *$\mathcal{C}'$ (i.e., $\mathcal{C}' \prec \mathcal{C}$). Moreover, $\mathcal{C}, \mathcal{C}'$ are* succinctly incomparable *if there is a problem L that separates $\mathcal{C}$ from $\mathcal{C}'$, and vice versa (i.e., $\mathcal{C} \npreceq \mathcal{C}'$ and $\mathcal{C}' \npreceq \mathcal{C}$).*

PARITY Representations and Completion

Note that (2) suggests a "pattern" for PARITY$_n$: The first part of the program (e.g., the "*not not*" rules in (2)) generates all possible strings of length $n-1$, the second part identifies the last bit to form an odd string of length n. Following the pattern, we can give a sequence of *exponential* size CP programs for PARITY, i.e., PARITY$\in$CP.

On the other hand, it is a textbook result that PARITY has polynomial size PF representations with classical connectives $\{\wedge, \vee, \neg\}$, i.e., PARITY$\in \mathsf{NC}^1/\mathsf{poly}$ (or Poly-PF). Remind that the size of a formula is the number of its connectives. E.g., $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$ represents PARITY$_2$, and the formula can be recursively applied for arbitrary PARITY$_n$. It is also well-known that PARITY$\notin \mathsf{AC}^0$, i.e., PARITY cannot be represented by a sequence of polynomial size circuits $\{C_n\}$ in which these circuits C_n have unbounded fan-in and constant depth.

Simply speaking, a sequence of polynomial size PF formulas $\{\phi_n\}$ represents an AC^0 language, if the number of variables occur in a *conjunction* or *disjunction* of ϕ_n is *unrestricted*, and the distance of the longest path from the root to a leaf in the tree structure of each ϕ_n is *fixed*. E.g., PARITY cannot be represented by a sequence of polynomial size CNFs $\{\psi_n\}$ since $\{\psi_n\}$ represents an AC^0 language. For more details about circuits, please see (Arora and Barak 2009; Jukna 2012).

The *completion* $Comp(\Pi)$ (Erdem and Lifschitz 2003) of a CP program Π consists of a set (or a conjunction) of PF formulas (we slightly abuse the connective $\equiv$): (i) $x \equiv \tilde{B}_1 \vee \tilde{B}_2 \vee \cdots \vee \tilde{B}_m$, where $x \leftarrow B_1, \ldots, x \leftarrow B_m$ are all rules in Π with head x, and each $\tilde{B}_i$ is the conjunction of rule elements in B_i with connective not replaced by $\neg$; (ii) $x \equiv \bot$, if x is not a head of any rule in Π; (iii) $\neg\tilde{B}$, if a rule $\bot \leftarrow B$ is in Π.

Proposition 2.1. *The completion $Comp(\Pi)$ of an arbitrary canonical program Π is a constant depth, unbounded fan-in circuit whose size is polynomially bounded by $|\Pi|$.*

It is well-known that an answer set of Π is also a model of its completion, but the inverse generally does not hold. E.g., the completion $\{x_1 \equiv \neg x_2 \vee (x_2 \wedge x_1), x_2 \equiv \neg x_1 \vee (x_1 \wedge x_2)\}$ of the PARITY$_2$ program:

$$\begin{array}{ll} x_1 \leftarrow not\ x_2, & x_2 \leftarrow not\ x_1, \\ x_1 \leftarrow x_2, not\ not\ x_1, & x_2 \leftarrow x_1, not\ not\ x_2, \end{array} \quad (4)$$

has an even string model 11, i.e., (4) is not *completion-equivalent*. In the next section, we will show how to simplify a PARITY program to be completion-equivalent.

3 Simplifying PARITY Programs

Let B be a finite set of rule elements built on signature $V = \{x_1, \ldots, x_n\}$. By $S(B)$ we denote the set $\{I \subseteq V : I \models B\}$. E.g., let $V = \{x_1, x_2, x_3, x_4\}$ and $B = \{x_2, not\ x_3, not\ not\ x_4\}$, then $S(B) = \{\{x_1, x_2, x_4\}, \{x_2, x_4\}\} = \{1101, 0101\}$. We say B is *consistent* if there is a set of variables I s.t. $I \models B$. We say B *covers* a variable $x \in V$ iff $x \in B$ or $not\ x \in B$ or $not\ not\ x \in B$. If B covers every variable in V then B *fully covers* V. E.g., $B = \{x_1, not\ x_2, not\ not\ x_3\}$ fully covers $V = \{x_1, x_2, x_3\}$. Clearly, B is consistent and fully covers V iff $S(B)$ contains a unique string.

In the following we assume that all rules $x \leftarrow B$ have consistent body B and $x \notin var(B)$. It is easy to see that the assumption does not affect the generality.

Lemma 3.1. *Let Π_n be a PARITY$_n$ program. If there is a rule $x \leftarrow B$ in Π_n s.t. $not\ not\ x \in B$ and $S(B)$ contains a unique even string, then dropping $x \leftarrow B$ from Π_n results in a PARITY$_n$ program Π'_n.*

E.g., by Lemma 3.1, (4) can be simplified to:

$$x_1 \leftarrow not\ x_2, \quad x_2 \leftarrow not\ x_1. \quad (5)$$

Lemma 3.2. *Let Π_n be a PARITY$_n$ program. If there is a rule $x \leftarrow B$ in Π_n s.t. $not\ not\ x \in B$ and $S(B)$ contains a unique odd string, then dropping $not\ not\ x$ from B results in a PARITY$_n$ program Π'_n.*

Standard Programs and the Main Theorem

A PARITY$_n$ program Π_n is *standard* if for each rule $x \leftarrow B \in \Pi_n$, $not\ not\ x \notin B$ whenever $S(B \cup \{x\})$ contains a unique string. E.g., (2) and (5) are standard programs. By Lemma 3.1 and 3.2, it is not hard to see:

Proposition 3.1. *Let Π_n be a PARITY$_n$ program. Then there is a standard PARITY$_n$ program Π'_n s.t. $|\Pi'_n| \leq |\Pi_n|$.*

Moreover, note that (2) and (5) are completion-equivalent PARITY programs. This is guaranteed by the following proposition:

Proposition 3.2. *Let Π_n be a standard PARITY$_n$ program. Then Π_n is equivalent to its completion $Comp(\Pi_n)$.*

The proof idea of Proposition 3.2 is that every standard PARITY$_n$ program Π_n can be equivalently rewritten to Π'_n by replacing each $x \in var(B)$ with $not\ not\ x$ for every rule body B in Π_n. By the Lin-Zhao Theorem (Lin and Zhao 2004) or the (generalized) *Fages Theorem* (Erdem and Lifschitz 2003; You, Yuan, and Zhang 2003), Π'_n is equivalent to its completion $Comp(\Pi'_n)$. And then the Proposition follows from the fact that $Comp(\Pi'_n) = Comp(\Pi_n)$.

Lemma 3.3 (Main Lemma)**.** *Let Π_n be a PARITY$_n$ program. Then there is a PARITY$_n$ program Π'_n s.t. Π'_n is equivalent to $Comp(\Pi'_n)$ and $|\Pi'_n| \leq |\Pi_n|$.*

Theorem 3.1 (PARITY$\notin$Poly-CP)**.** *PARITY has no polynomial size CP representations.*

Proof. Assume the contrary that there is a sequence of PARITY$_n$ programs $\{\Pi_n\}$ s.t. $|\Pi_n|$ is bounded by a polynomial $p(n)$. By Lemma 3.3, there is a sequence of completion-equivalent PARITY$_n$ programs $\{\Pi'_n\}$ in which $|\Pi'_n|$ is also bounded by the polynomial $p(n)$. By Proposition 2.1, $\{\Pi'_n\}$ represents a language in AC^0. This contradicts PARITY$\notin \mathsf{AC}^0$. □

Corollary 3.1. *PARITY separates PF from CP.*

Corollary 3.2. *Suppose $\mathsf{P} \nsubseteq \mathsf{NC}^1/\mathsf{poly}$. Then CP and PF are succinctly incomparable.*

4 Discussion and Conclusion

In this paper we show that the PARITY problem separates PF from CP, and the two formalisms are succinctly incomparable (assuming $\mathsf{P} \nsubseteq \mathsf{NC}^1/\mathsf{poly}$). Interestingly, our main result may at first appear counter-intuitive: the P-complete problem PATH has Poly-CP representations, while this does not hold for an "easy" problem PARITY. Actually, there is no contradiction. As noted in (Abiteboul, Hull, and Vianu 1995; Dantsin et al. 2001), a *complete* problem in a complexity class can be represented in a formalism $\mathcal{C}$, *does not* imply that *all* problems in that class can be represented in $\mathcal{C}$.

Generally speaking, the research of succinctness (Gogic et al. 1995; Coste-Marquis et al. 2004; Grohe and Schweikardt 2005; French et al. 2013) gives us a deeper understanding about KR formalisms, for it reveals their (in)abilities of concisely representing different problems under the condition that the encoded models are the *same*. This is particularly interesting if the formalisms are *equally expressive* and share the same reasoning complexity.

E.g., in addition to CP and NLP, programs with *cardinality constraints* and *choice rules* (CC, without classic negation $\neg$) (Simons, Niemelä, and Soininen 2002), *(simple) definite causal theories*[4] (S/DT) (Giunchiglia et al. 2004) and

[4] A theory is simple if each rule body is a conjunction of literals.

two-valued programs (TV) (Lifschitz 2012) are as expressive as PF and NP-complete for consistency checking. But they have a non-trivial succinctness picture, see Fig. 1.

Besides the theoretical interests, succinctness also tells us something like "which for what is the best" in choosing KR formalisms for a given application. E.g., one should choose ASP instead of SAT (or DT) if the application involves reasoning about PATH or Transitive Closure[5], because the former provides compact representations to avoid unnecessary overload. Recall that from the complexity viewpoint, even *one* extra variable may *double* the search space for intractable problems.

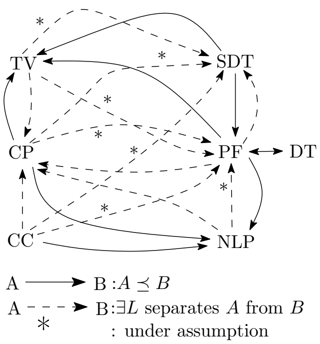

Figure 1: Succinctness Pic.

In future work we plan to establish the missing connections in Fig. 1, moreover, we will consider the succinctness of more expressive formalisms like programs with predicate symbols or higher-order atoms (Gebser, Schaub, and Thiele 2007), etc.

Acknowledgement

We are grateful to the anonymous reviewers for their valuable comments. Thanks to Shiguang Feng, Yan Zhang, Jiankun He, Guangrui Dang and Xiaolong Liang for their helpful discussions. The research was partially supported by NSFC Grant 61272059, MOE Grant 11JJD720020, NSSFC Grant 13&ZD186 and the Fundamental Research Funds for the Central Universities Grant 1409025.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Arora, S., and Barak, B. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12):92–103.

Coste-Marquis, S.; Lang, J.; Liberatore, P.; and Marquis, P. 2004. Expressive power and succinctness of propositional languages for preference representation. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning*, 203–212. AAAI Press.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33(3):374–425.

Erdem, E., and Lifschitz, V. 2003. Tight logic programs. *Theory and Practice of Logic Programming* 3(4):499–518.

French, T.; van der Hoek, W.; Iliev, P.; and Kooi, B. P. 2013. On the succinctness of some modal logics. *Artificial Intelligence* 197:56–85.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. Gringo: A new grounder for answer set programming. In *Proceedings of the 9th International Conference of Logic Programming and Nonmonotonic Reasoning (LNCS4483)*, 266–271. Springer.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; Turner, H.; and Lifschitz, J. L. V. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153(1-2):49–104.

Gogic, G.; Kautz, H. A.; Papadimitriou, C. H.; and Selman, B. 1995. The comparative linguistics of knowledge representation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 862–869. Morgan Kaufmann Publishers Inc.

Grohe, M., and Schweikardt, N. 2005. The succinctness of first-order logic on linear orders. *Logical Methods in Computer Science* 1(1).

Jukna, S. 2012. *Boolean Function Complexity - Advances and Frontiers*. Springer.

Lee, J.; Lifschitz, V.; and Yang, F. 2013. Action language $\mathcal{BC}$: Preliminary report. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 983–989. AAAI Press.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In *Proceedings of the 19th international joint conference on Artificial intelligence*, 503–508. Morgan Kaufmann Publishers Inc.

Lifschitz, V., and Razborov, A. 2006. Why are there so many loop formulas? *ACM Transactions on Computational Logic* 7(2):261–268.

Lifschitz, V. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence* 25:369–389.

Lifschitz, V. 2008. What is answer set programming? In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1594–1597. MIT Press.

Lifschitz, V. 2012. Two-valued logic programs. In *Technical Communications of the 28th International Conference on Logic Programming*, 259–266. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Lin, F., and Zhao, Y. 2004. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence* 157(1 - 2):115 – 137.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2):181–234.

You, J.-H.; Yuan, L.-Y.; and Zhang, M. 2003. On the equivalence between answer sets and models of completion for nested logic programs. In *Proceedings of the 18th international joint conference on Artificial intelligence*, 859–864. Morgan Kaufmann Publishers Inc.

[5]An NL-*complete* problem. It is believed that $\mathsf{NL} \not\subseteq \mathsf{NC}^1/\mathsf{Poly}$.