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
`fshyping, hsszxsg@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 P $\not\subseteq$ NC^1=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 P $\not\subseteq$ NC^1=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 :) (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 equivalent626 Tf PF.

2 Background

Canonical Programs

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

$$e := > j \ ? \ j \ x \ j \ not\ x \ j \ not\ not\ x$$

in which $>; ?$ 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

$$H \quad B \qquad (1)$$

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

$$\begin{array}{ll} x_1 \quad not\ not\ x_1; & x_3 \quad not\ x_1; not\ x_2; \\ x_2 \quad not\ not\ x_2; & x_3 \quad x_1;\ x_2: \end{array} \qquad (2)$$

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

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

$I \ j= \ >$ and $I \not\models \ ?$,

$I \ j= x$ iff $I \ j= not\ not\ x$ iff $x \ 2 \ I$,

$I \ j= not\ x$ iff $x \not2 \ I$.

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 , if I is closed under every rule in , i.e., for each rule $H \quad B \ 2$, $I \ j= H$ whenever $I \ j= B$. Let be a basic program and let $Cn($ $)$ denotes the *minimal* set (in terms of set inclusion) closed under , we say I is an *answer set* of if $I = Cn($ $)$. Note that a basic program has exactly one answer set.

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

$$x \quad not\ not\ x \qquad (3)$$

has two answer sets $;$ and fxg.

For a set of rule elements B, define $var(B) = fe \ 2 \ B : e$ is a variable$g$. E.g., $var(fx_1; not\ x_2; not\ not\ x_3g) = fx_1g$. The *signature* $sig($ $)$ of a program is the set of all involved variables in . By $Ans($ $)$ we denote the set of all answer sets of . E.g., if is (2), then $sig($ $) = fx_1; x_2; x_3g$ and $Ans($ $) = ffx_1; x_2; x_3g; fx_1g; fx_2g; fx_3gg$. The *size* $j \ j$ of is the number of its rules. As a convention, $_n$ refers to a program with signature $fx_1; : : : ; x_ng$, i.e., $sig($ $_n) = fx_1; : : : ; x_ng$.

It is easy to see that by using rules of the form (3) and appropriate *constraints* of the form $? \quad B$, it is easy to give an arbitrary set of answer sets over $sig($ $_n)$, 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 $f0; 1g$. A string w of length n (i.e., $w \ 2 \ f0; 1g^n$) defines a subset of variables $fx_1; : : : ; x_ng$. E.g., 1010 stands for $fx_1; x_3g$. 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) C (i.e., $L \ 2 \ C$), if there exists a* sequence *of programs $f \quad _ng$ ($n = 1; 2; : : :$) in C that* computes *L, i.e., for every string $w \ 2 \ f0; 1g^n$,*

$$w \ 2 \ L \quad , \quad w \ 2 \ Ans(\quad _n):$$

Say L has polay

The *completion* $Comp(\ \ )$ (Erdem and Lifschitz 2003) of a CP program consists of a set (or a conjunction) of PF formulas (we slightly abuse the connective): (i) $x \quad B_1 \_ B_2 \_ \quad \_ B_m$, where $x \quad B_1; : : : ; x \quad B_m$ are all rules in with head x, and each B_i is the conjunction of rule elements in B_i with connective *not* replaced by $:$; (ii) $x \quad ?$, if x is not a head of any rule in ; (iii) $: B$, if a rule $? \quad B$ is in .

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

It is well-known that an answer set of is also a model of its completion, but the inverse generally does not hold. E.g., the completion $f x_1 \quad : x_2 \_ (x_2 \wedge x_1); x_2 \quad : x_1 \_ (x_1 \wedge x_2) g$ of the PARITY$_2$ program:

$$\begin{array}{llll} x_1 & not\ x_2; & x_2 & not\ x_1; \\ x_1 & x_2; not\ not\ x_1; & x_2 & x_1; not\ not\ x_2; \end{array} \quad (4)$$

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.

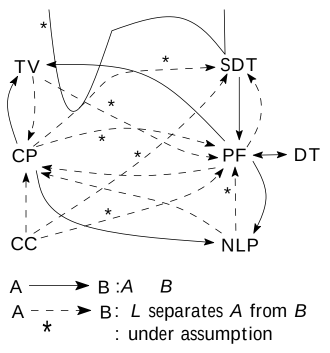

Figure 1: Succinctness Pic.

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:*