Deductive Systems for Logic Programs with Counting: Preliminary Report

Jorge Fandinno$^{1[0000-0002-3917-8717]}$ and
Vladimir Lifschitz$^{2[0000-0001-6051-7907]}$

^{1} University of Nebraska Omaha, Omaha, NE, USA `jfandinno@unomaha.edu`
^{2} University of Texas at Austin, Austin, TX, USA `lifschitzv@gmail.com`

Abstract. In answer set programming, two groups of rules are considered strongly equivalent if they have the same meaning in any context. Strong equivalence of two programs can be sometimes established by deriving rules of each program from rules of the other in an appropriate deductive system. This paper shows how to extend this method of proving strong equivalence to programs containing the counting aggregate.

Keywords: Answer Set Programming · Strong Equivalence · Deductive Systems · Aggregates.

1 Introduction

In answer set programming (ASP), two groups of rules are considered strongly equivalent if, informally speaking, they have the same meaning in any context [14]. If programs Π_1 and Π_2 are strongly equivalent then $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same stable models for any program Π. Properties of this equivalence relation are important because they can help us simplify parts of an ASP program without examining its other parts. More generally, they can guide us in the process of developing correct and efficient code.

Strong equivalence of two programs can be sometimes established by deriving rules of each program from rules of the other in an appropriate deductive system. Deriving rules involves rewriting them in the syntax of first-order logic. The possibility of such proofs has been demonstrated for the ASP language mini-GRINGO [4, 11, 13], and it was used in the design of a proof assistant for verifying strong equivalence [5, 9].

We are interested in extending this method of proving strong equivalence to ASP programs with aggregates, such as counting and summation [7, Section 3.1.12]. Procedures for representing rules with aggregates in the syntax of first-order logic have been proposed in several recent publications [1, 2, 12].

The last of these papers describes a deductive system that can be used to prove strong equivalence of programs in the language called mini-GRINGO with counting (MGC). But that system is too weak for reasoning about MGC rules that contain variables on the right-hand side of an aggregate atom. For instance, let A be the pair of rules

$$\begin{array}{l} p(a), \\ q(Y) \leftarrow \mathit{count}\{X : p(X) \wedge X \neq a\} = Y, \end{array}$$

and let B stand for

$$\begin{array}{l} p(a), \\ q(Y-1) \leftarrow \mathit{count}\{X : p(X)\} = Y. \end{array}$$

These pairs of rules are strongly equivalent to each other, but the deductive system mentioned above would not allow us to justify this claim.

We propose here a set of axioms for proving strong equivalence of programs with counting that is stronger than the axiom set from the previous publication [12]. After reviewing in Section 2 the language MGC and the translation τ^* that transforms MGC rules into first-order sentences, we define in Section 3 a deductive system of *here-and-there with counting (HTC)*. Any two MGC programs Π_1 and Π_2 such that $\tau^*\Pi_1$ and $\tau^*\Pi_2$ can be derived from each other in this deductive system are strongly equivalent. Furthermore, the sentences τ^*A and τ^*B, corresponding to the programs A and B above, are equivalent in HTC (Section 4).

The system HTC is not a first-order theory in the sense of classical logic, because some instances of the law of excluded middle $F \vee \neg F$ are not provable in it. This fact makes it difficult to automate reasoning in HTC, because existing work on automated reasoning deals for the most part with classical logic and its extensions. Lin [16] showed how to modify the straightforward representation of propositional rules by formulas in such a way that strong equivalence will correspond to equivalence of formulas in classical logic. In Section 5 we show that this method is applicable to programs with counting as well. To this end, we define a classical first-order theory HTC' and an additional syntactic transformation γ such that two sentences F_1, F_2 are equivalent in HTC if and only if γF_1 is equivalent to γF_2 in HTC'. It follows that if the formula $\gamma\tau^*\Pi_1 \leftrightarrow \gamma\tau^*\Pi_2$ can be derived from the axioms of HTC' in classical first-order logic then Π_1 is strongly equivalent to Π_2.

2 Background

2.1 Programs

The syntax of mini-GRINGO with counting is defined as follows.[3] We assume that three countably infinite sets of symbols are selected: *numerals*, *symbolic constants*, and *variables*. We assume that a 1-1 correspondence between numerals and integers is chosen; the numeral corresponding to an integer n is denoted by $\overline{n}$. (In examples of programs, we sometimes drop overlines in numerals.)

[3] The description below differs slightly from the original publication on MGC [12]: the absolute value symbol $|\,|$ is allowed in the definition of a term, and the symbols *inf* and *sup* are not included.

Precomputed terms are numerals and symbolic constants. We assume that a total order on the set of precomputed terms is selected such that numerals are contiguous (no symbolic constants between numerals) and are ordered in the standard way. MGC terms are formed from precomputed terms and variables using the unary operation symbol $|\,|$ and the binary operation symbols

$$+ \quad - \quad \times \quad / \quad \backslash \quad ..$$

An MGC *atom* is a symbolic constant optionally followed by a tuple of terms in parentheses. A *literal* is an MGC atom possibly preceded by one or two occurrences of *not*. A *comparison* is an expression of the form $t_1 \prec t_2$, where t_1, t_2 are mini-GRINGO terms, and $\prec$ is $=$ or one of the comparison symbols

$$\neq \quad < \quad > \quad \leq \quad \geq \tag{1}$$

An *aggregate element* is a pair $\mathbf{X} : \mathbf{L}$, where $\mathbf{X}$ is a tuple of distinct variables, and $\mathbf{L}$ is a conjunction of literals and comparisons such that every member of $\mathbf{X}$ occurs in $\mathbf{L}$. An *aggregate atom* is an expression of one of the forms

$$\mathit{count}\{E\} \geq t, \;\; \mathit{count}\{E\} \leq t, \tag{2}$$

where E is an aggregate element, and t is a term that does not contain the interval symbol (..). The conjunction of aggregate atoms (2) can be written as $\mathit{count}\{E\} = t$.

A *rule* is an expression of the form

$$\mathit{Head} \leftarrow \mathit{Body}, \tag{3}$$

where

- *Body* is a conjunction (possibly empty) of literals, comparisons, and aggregate atoms, and
- *Head* is either an atom (then (3) is a *basic rule*), or an atom in braces (then (3) is a *choice rule*), or empty (then (3) is a *constraint*).

A variable that occurs in a rule R is *local* in R if each of its occurrences is within an aggregate element, and *global* otherwise. A rule is *pure* if, for every aggregate element $\mathbf{X} : \mathbf{L}$ in its body, all variables in the tuple $\mathbf{X}$ are local. For example, the rule

$$q(Y) \leftarrow \mathit{count}\{X : p(X)\} = Y \wedge X > 0$$

is not pure, because X is global.

In mini-GRINGO with counting, a *program* is a finite set of pure rules.

2.2 Stable models and strong equivalence

An atom $p(\mathbf{t})$ is *precomputed* if all members of the tuple $\mathbf{t}$ are precomputed terms. The semantics of MGC is based on an operator, called τ, which transforms pure rules into infinitary propositional formulas formed from precomputed atoms [12, Section 5]. For example, the rule

$$q \leftarrow \mathit{count}\{X : p(X)\} \leq 5$$

is transformed by τ into the formula

$$\left(\bigwedge_{\Delta\,:\,|\Delta|>5} \neg \bigwedge_{x \in \Delta} p(x) \right) \to q,$$

where Δ ranges over finite sets of precomputed terms, and $|\Delta|$ stands for the cardinality of Δ. The result of applying τ to a program Π is defined as the conjunction of formulas τR for all rules R of Π.

Stable models of an MGC program Π are defined as stable models of $\tau\Pi$ in the sense of Truszczynski [17]. Thus stable models of programs are sets of precomputed atoms.

About programs Π_1 and Π_2 we say that they are *strongly equivalent* to each other if $\tau\Pi_1$ is strongly equivalent to $\tau\Pi_2$; in other words, if for every set Φ of infinitary propositional formulas formed from precomputed atoms, $\{\tau\Pi_1\} \cup \Phi$ and $\{\tau\Pi_2\} \cup \Phi$ have the same stable models. It is clear that if Π_1 is strongly equivalent to Π_2 then, for any program Π, $\Pi_1 \cup \Pi$ has the same stable models as $\Pi_2 \cup \Pi$ (take Φ to be $\{\tau\Pi\}$).

2.3 Representing MGC terms and atoms by formulas

In first-order formulas that represent programs we distinguish between terms of two sorts: the sort *general* and its subsort *integer*. General variables are meant to range over arbitrary precomputed terms, and we assume them to be the same as variables used in MGC programs. Integer variables are meant to range over numerals (or, equivalently, integers). In this paper, integer variables are represented by letters from the middle of the alphabet ($I, \dots, N$).

The two-sorted signature σ_0 includes

- all numerals as object constants of the sort *integer*;
- all symbolic constants as object constants of the sort *general*;
- the symbol $|\,|$ as a unary function constant; its argument and value have the sort *integer*;
- the symbols $+$, $-$ and $\times$ as binary function constants; their arguments and values have the sort *integer*;
- pairs p/n, where p is a symbolic constant and n is a nonnegative integer, as n-ary predicate constants; their arguments have the sort *general*;
- symbols (1) as binary predicate constants; their arguments have the sort *general.*

Note that the definition of σ_0 does not allow general variables in the scope of an arithmetic operation. For example, the MGC term $Y - \overline{1}$ is not a term over σ_0.

A formula of the form $(p/n)(\mathbf{t})$, where $\mathbf{t}$ is a tuple of terms, can be written also as $p(\mathbf{t})$. Thus precomputed atoms can be viewed as sentences over σ_0.

The set of values of an MGC term[4] t can be described by a formula over the signature σ_0 that contains a variable Z that does not occur in t [4, 13]. This formula, "Z is a value of t," is denoted by $val_t(Z)$. Its definition is recursive, and we reproduce here two of its clauses:

– if t is a precomputed term or a variable then $val_t(Z)$ is $Z = t$,
– if t is $t_1 \; op \; t_2$, where op is $+$, $-$, or $\times$ then $val_t(Z)$ is

$$\exists IJ(val_{t_1}(I) \wedge val_{t_2}(J) \wedge Z = I \; op \; J),$$

where I and J are integer variables that do not occur in t.

For example, $val_{Y-\overline{1}}(Z)$ is

$$\exists IJ(I = Y \wedge J = \overline{1} \wedge Z = I - J).$$

The translation τ^B transforms MGC atoms, literals and comparisons into formulas over the signature σ_0. (The superscript B conveys the idea that this translation expresses the meaning of expressions in *bodies* of rules.) For example, τ^B transforms $p(t)$ into the formula $\exists Z(val_t(Z) \wedge p(Z))$. The complete definition of τ^B can be found in other papers on strong equivalence [4, 13].

2.4 Representing aggregate expressions and rules

To represent aggregate expressions by first-order formulas, we need to extend the signature σ_0 [12, Section 7]. The signature σ_1 is obtained from σ_0 by adding all predicate constants of the forms

$$Atleast_F^{\mathbf{X};\mathbf{V}} \text{ and } Atmost_F^{\mathbf{X};\mathbf{V}} \tag{4}$$

where $\mathbf{X}$ and $\mathbf{V}$ are disjoint lists of distinct general variables, and F is a formula over σ_0 such that each of its free variables belongs to $\mathbf{X}$ or to $\mathbf{V}$. The number of arguments of each of constants (4) is greater by 1 than the length of $\mathbf{V}$; all arguments are of the sort *general*. If n is a positive integer then the formula $Atleast_F^{\mathbf{X};\mathbf{V}}(\mathbf{V}, \overline{n})$ is meant to express that F holds for at least n values of $\mathbf{X}$. The meaning of $Atmost_F^{\mathbf{X};\mathbf{V}}(\mathbf{V}, \overline{n})$ is similar: F holds for at most n values of $\mathbf{X}$.

For an aggregate atom of the form $count\{\mathbf{X} : \mathbf{L}\} \geq t$ in the body of a rule, the corresponding formula over σ_1 is

$$\exists Z \left(val_t(Z) \wedge Atleast^{\mathbf{X};\mathbf{V}}_{\exists \mathbf{W} \tau^B(\mathbf{L})}(\mathbf{V}, Z) \right),$$

where

[4] We talk about a *set* of values because an MGC term may contain the interval symbol. For instance, the values of the MGC term $\overline{1}..\overline{3}$ are $\overline{1}$, $\overline{2}$, and $\overline{3}$. On the other hand, the set of values of the term $a - \overline{1}$, where a is a symbolic constant, is empty.

- **V** is the list of global variables that occur in **L**, and
- **W** is the list of local variables that occur in **L** and are different from the members of **X**.

For example, the aggregate atom $\mathit{count}\{X : p(X)\} \geq Y$ is represented by the formula

$$\exists Z \left(Z = Y \wedge \mathit{Atleast}^{X;}_{\exists Z(Z=X \wedge p(Z))}(Z) \right)$$

(**V** and **W** are empty).

The formula representing $\mathit{count}\{\mathbf{X} : \mathbf{L}\} \leq t$ is formed in a similar way, with *Atmost* in place of *Atleast*.

Now we are ready to define the translation τ^*, which transforms pure rules into sentences over σ_1. It converts a basic rule

$$p(t) \leftarrow B_1 \wedge \cdots \wedge B_n$$

into the universal closure of the formula

$$B_1^* \wedge \cdots \wedge B_n^* \wedge \mathit{val}_t(Z) \rightarrow p(Z),$$

where B_i^* is $\tau^B(B_i)$ if B_i is a literal or comparison, and the formula representation of B_i formed as described above if B_i is an aggregate atom.

The definition of τ^* for pure rules of other forms can be found in the original paper on mini-GRINGO with counting [12, Sections 6 and 8]. For any program Π, $\tau^*\Pi$ stands for the conjunction of the sentences $\tau^* R$ for all rules R of Π.

2.5 Logic of here-and-there and standard interpretations

We are interested in deductive systems S with the following property:

for any programs Π_1 and Π_2,
if $\tau^*\Pi_1$ and $\tau^*\Pi_2$ can be derived from each other in S
then Π_1 is strongly equivalent to Π_2. (5)

Systems with property (5) cannot possibly sanction unlimited use of classical propositional logic. Consider, for instance, the one-rule programs

$$p \leftarrow \mathit{not}\ q \quad \text{and} \quad q \leftarrow \mathit{not}\ p.$$

They have different stable models, although the corresponding formulas $\neg q \rightarrow p$, $\neg p \rightarrow q$ have the same truth table.

This observation suggests that the study of subsystems of classical logic may be relevant. One such subsystem is first-order intuitionistic logic (with equality) adapted to the two-sorted signature σ_1. Intuitionistic logic does have property (5). Furthermore, this property is preserved if we add the axiom schema

$$F \vee (F \rightarrow G) \vee \neg G, \tag{6}$$

which characterizes the propositional logic of here-and-there [10].

The axiom schema

$$\exists X(F \to \forall X\, F) \tag{7}$$

(for a variable X of either sort) can be included without losing property (5) as well. It was introduced to extend the logic of here-and-there to a language with variables and quantifiers [15].

The axioms and inference rules discussed so far are abstract, in the sense that they are not related to any properties of the domains of variables (except that one is a subset of the other). To describe more specific axioms, we need the following definition. An interpretation of the signature σ_0 is *standard* if

– its domain of the sort *general* is the set of precomputed terms;
– its domain of the sort *integer* is the set of numerals;
– every object constant represents itself;
– the absolute value symbol and the binary function constants are interpreted as usual in arithmetic;
– predicate constants (1) are interpreted in accordance with the total order on precomputed terms chosen in the definition of MGC (Section 2.1).

Two standard interpretations of σ_0 can differ only by how they interpret the symbols p/n. If a sentence over σ_0 does not contain these symbols then it is either satisfied by all standard interpretations or is not satisfied by any of them.

Let *Std* be the set of all sentences over σ_0 that do not contain predicate symbols of the form p/n and are satisfied by standard interpretations. Property (5) will be preserved if we add any members of *Std* to the set of axioms. The set *Std* includes, for instance, the law of excluded middle $F \vee \neg F$ for every sentence F over σ_0 that does not contain symbols p/n. Other examples of formulas from *Std* are

$$\overline{2} \times \overline{2} = \overline{4}, \quad \forall N(N * N \geq \overline{0}), \quad t_1 \neq t_2,$$

where t_1, t_2 are distinct precomputed terms.

To reason about MGC programs, we need also axioms for *Atleast* and *Atmost*. A possible choice of such additional axioms is described in the next section.

3 Deductive system *HTC*

The deductive system *HTC* ("here-and-there with counting") operates with formulas of the signature σ_2, which is obtained from σ_1 (Section 2.4) by adding the predicate constants $Start_F^{\mathbf{X};\mathbf{V}}$, where $\mathbf{X}$ and $\mathbf{V}$ are disjoint lists of distinct general variables, and F is a formula over σ_0 such that each of its free variables belongs to $\mathbf{X}$ or to $\mathbf{V}$. The number of arguments of each of these constants is the combined length of $\mathbf{X}$ and $\mathbf{V}$ plus 1. The last argument is of the sort *integer*, and the other arguments are of the sort *general*. For any integer n, $Start_F^{\mathbf{X};\mathbf{V}}(\mathbf{X}, \mathbf{V}, \overline{n})$ is meant to express that if $n > 0$ then there exists a lexicographically increasing sequence $\mathbf{X}_1, \ldots \mathbf{X}_n$ of values satisfying F such that the first of them is $\mathbf{X}$.

3.1 Axioms of *HTC*

The axioms for *Start* define these predicates recursively:

$$\begin{array}{l}\forall \mathbf{X}\mathbf{V}N(N \leq \overline{0} \rightarrow \mathit{Start}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X},\mathbf{V},N)),\\ \forall \mathbf{X}\mathbf{V}(\mathit{Start}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X},\mathbf{V},\overline{1}) \leftrightarrow F),\\ \forall \mathbf{X}\mathbf{V}N(N > \overline{0} \rightarrow\\ \qquad (\mathit{Start}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X},\mathbf{V},N+\overline{1}) \leftrightarrow F \wedge \exists \mathbf{U}(\mathbf{X} < \mathbf{U} \wedge \mathit{Start}_F^{\mathbf{X};\mathbf{V}}(\mathbf{U},\mathbf{V},N)))).\end{array}$$

Here N is an integer variable, and $\mathbf{U}$ is a list of distinct general variables of the same length as $\mathbf{X}$ that is disjoint from both $\mathbf{X}$ and $\mathbf{V}$. The symbol $<$ in the last line denotes lexicographic order: $(X_1,\dots,X_m) < (U_1,\dots,U_m)$ stands for

$$\bigvee_{l=1}^{m}\left((X_l < U_l) \wedge \bigwedge_{k=1}^{l-1}(X_k = U_k)\right).$$

This set of axioms for *Start* will be denoted by D_0.

The set of axioms for *Atleast* and *Atmost*, denoted by D_1, defines these predicates in terms of *Start*:

$$\forall \mathbf{V}Y(\mathit{Atleast}_F^{\mathbf{X};\mathbf{V}}(\mathbf{V},Y) \leftrightarrow \exists \mathbf{X}N(\mathit{Start}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X},\mathbf{V},N) \wedge N \geq Y)), \tag{8}$$

$$\forall \mathbf{V}Y(\mathit{Atmost}_F^{\mathbf{X};\mathbf{V}}(\mathbf{V},Y) \leftrightarrow \forall \mathbf{X}N(\mathit{Start}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X},\mathbf{V},N) \rightarrow N \leq Y)). \tag{9}$$

In addition to the axioms listed above, we need the induction schema

$$F_{\overline{0}}^{N} \wedge \forall N\left(N \geq \overline{0} \wedge F \rightarrow F_{N+\overline{1}}^{N}\right) \rightarrow \forall N(N \geq \overline{0} \rightarrow F)$$

for all formulas F over σ_2. The set of the universal closures of its instances will be denoted by *Ind*.

The deductive system *HTC* is defined as first-order intuitionistic logic for the signature σ_2 extended by

- axiom schemas (6) and (7) for all formulas F, G, H over σ_2, and
- axioms *Std*, *Ind*, D_0 and D_1.

This deductive system has property (5):

Theorem 1. *For programs Π_1 and Π_2, if formula $\tau^*\Pi_1 \leftrightarrow \tau^*\Pi_2$ is provable in HTC then Π_1 and Π_2 are strongly equivalent.*

If the claim that Π_1 is strongly equivalent to Π_2 can be justified by the method from the previous publication [12, Theorem 3] then it can be justified by the theorem above as well. Furthermore, in Section 4 we show that *HTC* is sufficiently strong for proving the equivalence between τ^*A and τ^*B for the programs A and B from the introduction.

3.2 Some theorems of *HTC*

The characterization of *Atleast* and *Atmost* given by the axioms D_1 can be simplified, if we replace the variable Y by an integer variable:

Proposition 1. *The formulas*

$$\forall \mathbf{V} N(\mathit{Atleast}_F^{\mathbf{X};\mathbf{V}}(\mathbf{V}, N) \leftrightarrow \exists \mathbf{X}\, \mathit{Start}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X}, \mathbf{V}, N)) \tag{10}$$

and

$$\forall \mathbf{V} N(\mathit{Atmost}_F^{\mathbf{X};\mathbf{V}}(\mathbf{V}, N) \leftrightarrow \neg \mathit{Atleast}_F^{\mathbf{X};\mathbf{V}}(\mathbf{V}, N + \overline{1})) \tag{11}$$

are provable in HTC.

Proposition 2. *The formulas*

$$\forall \mathbf{V} N(N \leq \overline{0} \to \mathit{Atleast}_F^{\mathbf{X};\mathbf{V}}(\mathbf{V}, N)), \tag{12}$$

$$\forall \mathbf{V}(\mathit{Atleast}_F^{\mathbf{X};\mathbf{V}}(\mathbf{V}, \overline{1}) \leftrightarrow \exists \mathbf{X}\, F), \tag{13}$$

$$\forall \mathbf{X}(F \to G) \to \forall \mathbf{X} \mathbf{V} N(\mathit{Start}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X}, \mathbf{V}, N) \to \mathit{Start}_G^{\mathbf{X};\mathbf{V}}(\mathbf{X}, \mathbf{V}, N)), \tag{14}$$

$$\forall \mathbf{Z} \mathbf{V} N(\mathit{Start}_F^{\mathbf{X};\mathbf{V}}(\mathbf{Z}, \mathbf{V}, N) \wedge N > \overline{0} \to F_{\mathbf{Z}}^{\mathbf{X}}) \tag{15}$$

are provable in HTC.

An expression of the form $\mathit{Exactly}_F^{\mathbf{X};\mathbf{V}}(\mathbf{t}, t)$ is shorthand for the conjunction

$$\mathit{Atleast}_F^{\mathbf{X};\mathbf{V}}(\mathbf{t}, t) \wedge \mathit{Atmost}_F^{\mathbf{X};\mathbf{V}}(\mathbf{t}, t)$$

($\mathbf{t}$ is a tuple of terms; t is a term). By (11), $\mathit{Exactly}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X}, N)$ is equivalent to

$$\mathit{Atleast}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X}, N) \wedge \neg \mathit{Atleast}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X}, N + \overline{1}).$$

Proposition 3. *The formulas*

$$\forall \mathbf{X} Y(\mathit{Exactly}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X}, Y) \to \exists N(Y = N \wedge N \geq \overline{0})) \tag{16}$$

and

$$\forall \mathbf{X}(F \leftrightarrow G) \to \forall \mathbf{X} Y(\mathit{Exactly}_F^{\mathbf{X};\mathbf{V}}(\mathbf{X}, Y) \leftrightarrow \mathit{Exactly}_G^{\mathbf{X};\mathbf{V}}(\mathbf{X}, Y)) \tag{17}$$

are provable in HTC.

4 An example of reasoning about programs

We outline here a proof of the equivalence $\tau^*A \leftrightarrow \tau^*B$, for the programs A and B from the introduction, in the deductive system HTC.

The translation τ^* transforms program A into the conjunction of the formulas

$$\forall Z(Z = a \to p(Z)) \tag{18}$$

and

$$\begin{array}{l}\forall YZ(\exists Z_1(Z_1 = Y \wedge \mathit{Atleast}^{X;}_{F}(Z_1)) \wedge \\ \quad \exists Z_2(Z_2 = Y \wedge \mathit{Atmost}^{X;}_{F}(Z_2)) \wedge Z = Y \to q(Z)),\end{array} \tag{19}$$

where F stands for $\tau^B(p(a) \wedge X \neq a)$. Formula (18) is equivalent to $p(a)$, and (19) is equivalent to

$$\forall Y(\mathit{Atleast}^{X;}_{F}(Y) \wedge \mathit{Atmost}^{X;}_{F}(Y) \to q(Y)). \tag{20}$$

The antecedent of this implication can be written as $\mathit{Exactly}^{X;}_{F}(Y)$. By (16), it follows that the variable Y can be replaced by the integer variable N. Furthermore, by (17), formula (20) can be further rewritten as

$$\forall N(\mathit{Exactly}^{X;}_{p(a)\wedge X\neq a}(N) \to q(N)), \tag{21}$$

because F is equivalent to $p(a) \wedge X \neq a$.

The result of applying τ^* to B is the conjunction of (18) and

$$\begin{array}{r}\forall YZ(\exists Z_1(Z_1 = Y \wedge \mathit{Atleast}^{X;}_{G}(Z_1)) \wedge \exists Z_2(Z_2 = Y \wedge \mathit{Atmost}^{X;}_{G}(Z_2)) \wedge \\ \exists IJ(I = Y \wedge J = \overline{1} \wedge Z = I + J) \to q(Z)),\end{array}$$

where G stands for $\tau^B(p(X))$. This formula can be equivalently rewritten as

$$\forall I(\mathit{Exactly}^{X;}_{G}(I + \overline{1}) \to q(I))$$

and further as

$$\forall I(\mathit{Exactly}^{X;}_{p(X)}(I + \overline{1}) \to q(I)), \tag{22}$$

because G is equivalent to $p(X)$.

Thus the claim that τ^*A is equivalent to τ^*B will be proved if we prove

$$p(a) \to \forall N(\mathit{Exactly}^{X;}_{p(X)\wedge X\neq a}(N + \overline{1}) \leftrightarrow \mathit{Exactly}^{X;}_{p(X)}(N)).$$

This formula is clearly a consequence of

$$p(a) \to \forall N(\mathit{Atleast}^{X;}_{p(X)\wedge X\neq a}(N + \overline{1}) \leftrightarrow \mathit{Atleast}^{X;}_{p(X)}(N)). \tag{23}$$

We will show now that (23) can be derived from three lemmas:

$$N > \overline{0} \wedge X > a \wedge \mathit{Start}^{X;}_{p(X)}(X, N) \to \mathit{Start}^{X;}_{p(X)\wedge X\neq a}(X, N), \tag{24}$$

$$N > \overline{0} \wedge X \neq a \wedge \mathit{Start}^{X;}_{p(X)}(X, N + \overline{1}) \to \mathit{Start}^{X;}_{p(X)\wedge X\neq a}(X, N), \tag{25}$$

$$N > \overline{0} \wedge X < a \wedge p(a) \wedge \mathit{Start}^{X;}_{p(X)\wedge X\neq a}(X, N) \to \mathit{Start}^{X;}_{p(X)}(X, N + \overline{1}). \tag{26}$$

Assume $p(a)$; our goal is to show that

$$Atleast^{X;}_{p(X)\wedge X\neq a}(N+\overline{1}) \leftrightarrow Atleast^{X;}_{p(X)}(N).$$

We consider three cases, given by the Std axiom $\forall N(N<\overline{0} \vee N=\overline{0} \vee N>\overline{0})$.

If $N<\overline{0}$ then both sides of the equivalence are true by (12). If $N=\overline{0}$ then the right-hand side is true by (12), and the left-hand side follows from $p(a)$ by (13). Assume that $N>\overline{0}$.

Left-to-right: assume $Atleast^{X;}_{p(X)}(N+\overline{1})$. By (10), there exists X such that

$$Start^{X;}_{p(X)}(X, N+\overline{1}). \tag{27}$$

Case 1: $X=a$, so that $Start^{X;}_{p(X)}(a, N+\overline{1})$. By D_0,

$$p(a) \wedge \exists U(a<U \wedge Start^{X;}_{p(X)}(U,N)).$$

Take U such that $a<U$ and $Start^{X;}_{p(X)}(U,N)$. By (24), it follows that

$$Start^{X;}_{p(X)\wedge X\neq a}(U,N).$$

Then $Atleast^{X;}_{p(X)\wedge X\neq a}(N)$ by (10). *Case 2:* $X\neq a$. By (27) and (25),

$$Start^{X;}_{p(X)\wedge X\neq a}(X,N).$$

By (10), it follows that $Atleast^{X;}_{p(X)\wedge X\neq a}(N)$.

Right-to-left: assume $Atleast^{X;}_{p(X)\wedge X\neq a}(N)$. Then, for some X,

$$Start^{X;}_{p(X)\wedge X\neq a}(X,N) \tag{28}$$

by (10), and consequently $Start^{X;}_{p(X)}(X,N)$ by (14). *Case 1:* $X>a$. Then

$$p(a) \wedge \exists U(a<U \wedge Start^{X;}_{p(X)}(U,N))$$

(take U to be X). By D_0, we conclude that $Start^{X;}_{p(X)}(a, N+\overline{1})$. Then $Atleast^{X;}_{p(X)}(N+\overline{1})$ follows by (10). *Case 2:* $X\leq a$. From (28) and (15), $X\neq a$, so that $X<a$. From (28) and (26), $Start^{X;}_{p(X)}(X, N+\overline{1})$; $Atleast^{X;}_{p(X)}(N+\overline{1})$ follows by (10).

Proofs of lemmas (24)–(26) use induction *Ind*, and they are not included here because of the limit on the length of conference submissions.

5 Deductive system *HTC′*

In this section, we show how an additional syntactic transformation γ allows us to replace *HTC* by a classical first-order theory.

The signature σ_2' is obtained from the signature σ_2 (Section 3.1) by adding, for every predicate symbol p other than comparison symbols (1), a new predicate symbol p' of the same arity. The formula $\forall\mathbf{X}(p(\mathbf{X}) \to p'(\mathbf{X}))$, where $\mathbf{X}$ is a tuple of distinct general variables, is denoted by $\mathcal{A}(p)$. The set of all formulas $\mathcal{A}(p)$ is denoted by $\mathcal{A}$.

For any formula F over the signature σ_2, by F' we denote the formula over σ_2' obtained from F by replacing every occurrence of every predicate symbol p other than comparison symbols by p'. The translation γ, which relates the logic of here-and-there to classical logic, maps formulas over σ_2 to formulas over σ_2'. It is defined recursively:

- $\gamma F = F$ if F is atomic,
- $\gamma(\neg F) = \neg F'$,
- $\gamma(F \wedge G) = \gamma F \wedge \gamma G$,
- $\gamma(F \vee G) = \gamma F \vee \gamma G$,
- $\gamma(F \to G) = (\gamma F \to \gamma G) \wedge (F' \to G')$,
- $\gamma(\forall X\, F) = \forall X\, \gamma F$,
- $\gamma(\exists X\, F) = \exists X\, \gamma F$.

To apply γ to a set of formulas means to apply γ to each of its members.

By HTC' we denote the classical first-order theory over the signature σ_2' with the axioms $\mathcal{A}$, $\gamma(Ind)$, Std, γD_0 and γD_1.

Theorem 2. *A sentence $F \leftrightarrow G$ over the signature σ_2 is provable in HTC iff $\gamma F \leftrightarrow \gamma G$ is provable in HTC'.*

From Theorems 1 and 2 we conclude that MGC programs Π_1 and Π_2 are strongly equivalent if the formula $\gamma\tau^*\Pi_1 \leftrightarrow \gamma\tau^*\Pi_2$ is provable in HTC'.

6 Conclusion

In this paper we described the deductive system HTC and argued that strong equivalence of two programs with counting can be established in many cases by proving the equivalence of the corresponding first-order sentences in that system. The question whether HTC is complete for strong equivalence, that is to say, whether $\tau^*\Pi_1 \leftrightarrow \tau^*\Pi_2$ is provable in HTC for all pairs Π_1, Π_2 of strongly equivalent MGC programs, is an open problem.

We described also a modification HTC' of HTC, which is a first-order theory in the sense of classical logic such that sentences F_1, F_2 are equivalent in HTC if and only if the sentences γF_1, γF_2 are equivalent in HTC'. This fact suggests that it may be possible to use theorem provers for classical theories to verify strong equivalence of MGC programs.

A translation closely related to τ^* is used in ANTHEM [6] to verify another kind of equivalence of mini-GRINGO programs—equivalence with respect to a user guide [3, 8]. We plan to extend work on user guides to programs with counting.

Finally, we would like to investigate the possibility of extending the deductive systems described in this paper to aggregates other than counting.

References

1. Fandinno, J., Hansen, Z., Lierler, Y.: Axiomatization of aggregates in answer set programming. In: Proceedings of the AAAI Conference on Artificial Intelligence (2022)
2. Fandinno, J., Hansen, Z.: Recursive aggregates as intensional functions. In: Proceedings of the Workshops co-located with the 39th International Conference on Logic Programming (2023)
3. Fandinno, J., Hansen, Z., Lierler, Y., Lifschitz, V., Temple, N.: External behavior of a logic program and verification of refactoring. Theory and Practice of Logic Programming (2023)
4. Fandinno, J., Lifschitz, V.: Omega-completeness of the logic of here-and-there and strong equivalence of logic programs. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (2023)
5. Fandinno, J., Lifschitz, V.: On Heuer's procedure for verifying strong equivalence. In: Proceedings of European Conference on Logics in Artificial Intelligence (2023)
6. Fandinno, J., Lifschitz, V., Lühne, P., Schaub, T.: Verifying tight logic programs with Anthem and Vampire. Theory and Practice of Logic Programming **20** (2020)
7. Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S.: Potassco User Guide (2019), available at https://github.com/potassco/guide/releases/
8. Hansen, Z.: Anthem-p2p: Automatically verifying the equivalent external behavior of ASP programs. In: Electronic Proceedings in Theoretical Computer Science. vol. 385 (2023)
9. Heuer, J.: Automated verification of equivalence properties in advanced logic programs (2020), Bachelor Thesis, University of Potsdam
10. Hosoi, T.: The axiomatization of the intermediate propositional systems S_n of Gödel. Journal of the Faculty of Science of the University of Tokyo **13**, 183–187 (1966)
11. Lifschitz, V.: Here and there with arithmetic. Theory and Practice of Logic Programming (2021)
12. Lifschitz, V.: Strong equivalence of logic programs with counting. Theory and Practice of Logic Programming **22** (2022)
13. Lifschitz, V., Lühne, P., Schaub, T.: Verifying strong equivalence of programs in the input language of gringo. In: Proceedings of the 15th International Conference on Logic Programming and Non-monotonic Reasoning (2019)
14. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic **2**, 526–541 (2001)
15. Lifschitz, V., Pearce, D., Valverde, A.: A characterization of strong equivalence for logic programs with variables. In: Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). pp. 188–200 (2007)
16. Lin, F.: Reducing strong equivalence of logic programs to entailment in classical propositional logic. In: Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR). pp. 170–176 (2002)
17. Truszczynski, M.: Connecting first-order ASP and the logic FO(ID) through reducts. In: Erdem, E., Lee, J., Lierler, Y., Pearce, D. (eds.) Correct Reasoning: Essays on Logic-Based AI in Honor of Vladimir Lifschitz, pp. 543–559. Springer (2012)