Strong Equivalence
of Logic Programs with Counting

VLADIMIR LIFSCHITZ

University of Texas at Austin, USA
(*e-mail:* `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. In some cases, strong equivalence of programs in the input language of the grounder GRINGO can be established by deriving rules of each program from rules of the other. The possibility of such proofs has been demonstrated for a subset of that language that includes comparisons, arithmetic operations, and simple choice rules, but not aggregates. This method is extended here to a class of programs in which some uses of the `#count` aggregate are allowed.

KEYWORDS: aggregates, answer set programming, strong equivalence

1 Introduction

In answer set programming (Marek and Truszczynski 1999; Niemelä 1999; Gelfond and Kahl 2014; Lifschitz 2019), two groups of rules are considered strongly equivalent if, informally speaking, they have the same meaning in any context (Lifschitz *et al.* 2001). We are interested in proving strong equivalence of programs in the input language of the grounder GRINGO (Gebser *et al.* 2019) by deriving rules of each program from rules of the other. The possibility of such proofs has been demonstrated for the subset of that language called mini-GRINGO (Lifschitz *et al.* 2019; Lifschitz 2021). Programs allowed in that subset may include comparisons, arithmetic operations, and simple choice rules, but not aggregates.

The process of proving strong equivalence uses the translation τ^* (Lifschitz *et al.* 2019, Section 6) that transforms mini-GRINGO rules into first-order formulas with two sorts of variables – for numerals and for arbitrary precomputed terms. If two mini-GRINGO programs, rewritten as sets of sentences, are equivalent in the deductive system *HTA* ("here-and-there with arithmetic"), then they are strongly equivalent (Lifschitz 2021, Section 4).

In this paper, τ^* is extended to a superset of mini-GRINGO in which the `#count` aggregate can be used in a limited way. The study of the strong equivalence relation between GRINGO programs with counting and other aggregates is important because these constructs are widely used in answer set programming, and because some properties of this relation in the presence of aggregates may seem counterintuitive. For instance, the rule

$$\texttt{q :- \#count\{X : p(X)\} >= Y, Y = 1.} \qquad (1)$$

CrossMark

is strongly equivalent to each of the simpler rules

$$\texttt{q :- \#count\{X : p(X)\} >= 1.} \tag{2}$$

and

$$\texttt{q :- p(X).} \tag{3}$$

—as could be expected. But the rule

$$\texttt{q :- \#count\{X : p(X)\} = Y, Y >= 1.} \tag{4}$$

is not strongly equivalent to (2) and (3). Indeed, adding the rules

$$\begin{array}{l} \texttt{p(a).} \\ \texttt{p(b) :- q.} \end{array} \tag{5}$$

to (4) gives a program without stable models.[1]

The syntax and semantics of mini-GRINGO rules are reviewed in Sections 2 and 3. The semantics is defined by transforming rules into infinite sets of propositional formulas (Lifschitz *et al.* 2019, Section 3) and then appealing to the propositional stable model semantics (Lifschitz 2019, Section 5.2). Adding the #count aggregate, described in Sections 4 and 5, involves stable models of *infinitary* propositional formulas (Truszczynski 2012, Section 2), in the spirit of the approach of Gebser *et al.* (2015). The translation τ^* is reviewed in Section 6 and extended to mini-GRINGO with counting in Sections 7 and 8. After providing additional background information in Section 9, we state in Section 10 three theorems expressing properties of the translation and show how the strong equivalence of rules (1)–(3) can be proved by deriving them from each other. Proofs of the theorems are given in Section 11.

2 Review: syntax of mini-gringo

The description of mini-GRINGO programs below uses "abstract syntax," which disregards some details related to representing programs by strings of ASCII characters. 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 $\overline{n}$. *Precomputed terms* are numerals, symbolic constants, and the symbols *inf*, *sup*. We assume that a total order on the set of precomputed terms is selected, with the least element *inf* and the greatest element *sup*, so that numerals are contiguous and ordered in the standard way.

Terms allowed in a mini-GRINGO program are formed from precomputed terms and variables using the binary operation symbols

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

An *atom* is a symbolic constant optionally followed by a tuple of terms in parentheses. A *literal* is an atom possibly preceded by one or two occurrences of *not*. A *comparison*

---

[1] This is a modification of an example due to Gelfond and Zhang (2019). This example shows that the use of functional notation for #count is sometimes misleading. The correspondence between sets and their cardinalities is a total function classically, but not intuitionistically. We return to this question in the discussion of related work (Section 12).

is an expression of the form $t_1 \prec t_2$, where t_1, t_2 are mini-GRINGO terms and $\prec$ is one of the six comparison symbols

$$= \quad \neq \quad < \quad > \quad \leq \quad \geq. \tag{6}$$

A *mini-GRINGO rule* is an expression of the form

$$Head \leftarrow Body, \tag{7}$$

where

- *Body* is a conjunction (possibly empty) of literals and comparisons, and
- *Head* is either an atom (then (7) is a *basic rule*), or an atom in braces (then (7) is a *choice rule*), or empty (then (7) is a *constraint*).

A *mini-GRINGO program* is a finite set of mini-GRINGO rules.

3 Review: Semantics of mini-gringo

The semantics of ground terms is defined by assigning to every ground term t the finite set $[t]$ of its *values* (Lifschitz *et al.* 2019, Section 3). Values of a ground term are precomputed terms. For instance,

$$[\overline{2/2}] = \{\overline{1}\}, \ [\overline{2/0}] = \emptyset, \ [\overline{0 \,..\, 2}] = \{\overline{0}, \overline{1}, \overline{2}\}.$$

If a term is interval-free (that is, does not contain ..), then it has at most one value. For any ground terms $t_1, \ldots, t_n$, by $[t_1, \ldots, t_n]$, we denote the set of tuples $r_1, \ldots, r_n$ such that $r_1 \in [t_1], \ldots, r_n \in [t_n]$.

Stable models of a mini-GRINGO program are defined as stable models of the set of propositional formulas obtained from it by applying a syntactic transformation denoted by τ (Lifschitz *et al.* 2019, Section 3). These propositional formulas are built from *precomputed atoms*—atoms $p(\mathbf{r})$ such that members of the tuple $\mathbf{r}$ are precomputed terms. Thus, every stable model is a set of precomputed atoms.

The transformation τ is defined as follows. For any ground atom $p(\mathbf{t})$,

- $\tau(p(\mathbf{t}))$ is $\bigvee_{\mathbf{r} \in [\mathbf{t}]} p(\mathbf{r})$,
- $\tau(not\ p(\mathbf{t}))$ is $\bigvee_{\mathbf{r} \in [\mathbf{t}]} \neg p(\mathbf{r})$, and
- $\tau(not\ not\ p(\mathbf{t}))$ is $\bigvee_{\mathbf{r} \in [\mathbf{t}]} \neg\neg p(\mathbf{r})$.

For any ground comparison $t_1 \prec t_2$, $\tau(t_1 \prec t_2)$ is $\top$ if the relation $\prec$ holds between some r_1 from $[t_1]$ and some r_2 from $[t_2]$, and $\bot$ otherwise. The result of applying τ to a conjunction $B_1 \wedge B_2 \wedge \cdots$ is $\tau(B_1) \wedge \tau(B_2) \wedge \cdots$. If R is a ground basic rule $p(\mathbf{t}) \leftarrow Body$, then $\tau(R)$ is the propositional formula

$$\tau(Body) \rightarrow \bigwedge_{\mathbf{r} \in [\mathbf{t}]} p(\mathbf{r}). \tag{8}$$

If R is a ground choice rule $\{p(\mathbf{t})\} \leftarrow Body$ then $\tau(R)$ is the propositional formula

$$\tau(Body) \rightarrow \bigwedge_{\mathbf{r} \in [\mathbf{t}]} (p(\mathbf{r}) \vee \neg p(\mathbf{r})). \tag{9}$$

If R is a ground constraint $\leftarrow Body$ then $\tau(R)$ is

$$\neg\tau(Body). \tag{10}$$

For any mini-GRINGO program Π, $\tau(\Pi)$ is the set of propositional formulas $\tau(R)$ for all ground rules R that can be obtained from rules of Π by substituting precomputed terms for variables.

For example, substituting a precomputed term r for X in the choice rule

$$\{p(\overline{2})\} \leftarrow p(X), \tag{11}$$

gives the ground rule $\{p(\overline{2})\} \leftarrow p(r)$, so that τ transforms (11) into the set of all formulas of the form

$$p(r) \rightarrow p(\overline{2}) \vee \neg p(\overline{2}).$$

4 Mini-gringo with counting: syntax

We extend the class of mini-GRINGO rules as follows. 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 $\mathbf{L}$. In mini-GRINGO with counting, the body of a rule is allowed to contain, besides literals and comparisons, *aggregate atoms* of the forms

$$count\{E\} \geq t, \ count\{E\} \leq t, \tag{12}$$

where E is an aggregate element, and t is an interval-free term. The conjunction of aggregate atoms (12) can be written as $count\{E\} = t$.

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, all rules that do not contain aggregate elements are pure. The rules

$$q \leftarrow count\{X : p(X,Y)\} \leq \overline{2}, \tag{13}$$

and

$$q \leftarrow count\{X : p(X,Y)\} \leq \overline{2} \wedge Y = \overline{1} .. \overline{10}, \tag{14}$$

are pure, because X is local in each of them. The rule

$$q \leftarrow count\{X : p(X,Y)\} \leq \overline{2} \wedge X = \overline{1} .. \overline{10}, \tag{15}$$

is not pure, because X is global.[2]

A *program in mini-GRINGO with counting*, or an MGC *program*, is a finite set of pure rules. Allowing nonpure rules in a program would necessitate making the semantics more complicated; see Footnote 3.

An expression of the form

$$\overline{m}\,\{\mathbf{X} : A\}\,\overline{n} \leftarrow Body,$$

where $\mathbf{X}$ is a tuple of distinct variables, A is an atom, and $Body$ is a conjunction of literals, comparisons and aggregate atoms, can be used as shorthand for the group of three rules:

$$\{A\} \leftarrow Body,$$
$$\leftarrow Body, count\{\mathbf{X} : A\} \leq \overline{m-1},$$
$$\leftarrow Body, count\{\mathbf{X} : A\} \geq \overline{n+1}.$$

---

[2] In response to rules like this, the current version of GRINGO produces a warning message: `global variable in tuple of aggregate element`.

5 Mini-gringo with counting: Semantics

To define the semantics of MGC programs, we will extend the definition of τ (Section 3) to aggregate atoms (12) such that t is a ground term. Since t is interval-free, $[t]$ is either a singleton or empty. If $[t]$ is a singleton $\{c\}$, then $\tau(count\{\mathbf{X} : \mathbf{L}\} \geq t)$ is defined as the infinite disjunction

$$\bigvee_{\Delta \,:\, \overline{|\Delta|} \geq c} \bigwedge_{\mathbf{x} \in \Delta} \bigvee_{\mathbf{w}} \tau\left(\mathbf{L}_{\mathbf{x},\,\mathbf{w}}^{\mathbf{X},\mathbf{W}}\right), \tag{16}$$

and $\tau(count\{\mathbf{X} : \mathbf{L}\} \leq t)$ as the infinite conjunction

$$\bigwedge_{\Delta \,:\, \overline{|\Delta|} > c} \neg \bigwedge_{\mathbf{x} \in \Delta} \bigvee_{\mathbf{w}} \tau\left(\mathbf{L}_{\mathbf{x},\,\mathbf{w}}^{\mathbf{X},\mathbf{W}}\right), \tag{17}$$

where

- Δ ranges over finite sets of tuples of precomputed terms of the same length as $\mathbf{X}$;
- $\mathbf{W}$ is the list of variables that occur in $\mathbf{L}$ but do not belong to $\mathbf{X}$;
- $\mathbf{w}$ ranges over tuples of precomputed terms of the same length as $\mathbf{W}$;
- the expression $\mathbf{L}_{\mathbf{x},\,\mathbf{w}}^{\mathbf{X},\mathbf{W}}$ denotes the result of substituting $\mathbf{x}$, $\mathbf{w}$ for all occurrences of $\mathbf{X}$, $\mathbf{W}$ in $\mathbf{L}$.

(If c is a numeral $\overline{n}$ then the inequalities $\overline{|\Delta|} \geq c$, $\overline{|\Delta|} > c$ in these formulas can be written as $|\Delta| \geq n$, $|\Delta| > n$.) If $[t]$ is empty, then we define

$$\tau(count\{\mathbf{X} : \mathbf{L}\} \geq t) = \tau(count\{\mathbf{X} : \mathbf{L}\} \leq t) = \bot.$$

For example, the result of applying τ to the body of rule (13) is

$$\bigwedge_{\Delta \,:\, |\Delta| > \overline{2}} \neg \bigwedge_{x \in \Delta} \bigvee_{w} p(x, w).$$

(In this case, $\mathbf{X}$ is X; $\mathbf{W}$ is Y; Δ ranges over sets of precomputed terms; w ranges over precomputed terms.) In application to the aggregate expression

$$count\{X : p(X, r)\} \leq \overline{2},$$

where r is a precomputed term, τ gives

$$\bigwedge_{\Delta \,:\, |\Delta| > \overline{2}} \neg \bigwedge_{x \in \Delta} p(x, r),$$

($\mathbf{W}$ is empty).

A rule is *closed* if it has no global variables. It is clear that substituting precomputed terms for all global variables in a pure rule is a closed pure rule.[3] For any mini-GRINGO program Π, $\tau(\Pi)$ stands for the conjunction of the formulas $\tau(R)$ over all closed rules R that can be obtained from rules of Π by such substitutions. Thus, $\tau(\Pi)$ is an infinitary propositional formula over the signature consisting of all precomputed atoms.

---

[3] In case of a rule that is not pure, substituting precomputed terms for global variables may transform an aggregate element in the body into an expression that is not an aggregate element. For instance, substituting $\overline{1}$ for X in rule (15) turns $X : p(X, Y)$ into the expression $\overline{1} : p(\overline{1}, Y)$, which is not allowed by the syntax of mini-GRINGO with counting.

For example, τ transforms rule (13) into

$$\bigwedge_{\Delta \,:\, |\Delta| > \overline{2}} \neg \bigwedge_{x \in \Delta} \bigvee_{w} p(x, w) \rightarrow q,$$

where Δ ranges over sets of precomputed terms, and w ranges over precomputed terms. Rule (14) becomes

$$\bigwedge_{r} \left(\left(\bigwedge_{\Delta \,:\, |\Delta| > \overline{2}} \neg \bigwedge_{x \in \Delta} p(x, r) \right) \wedge \tau(r = \overline{1}..\overline{10}) \rightarrow q \right),$$

where r ranges over precomputed terms, and Δ ranges over sets of precomputed terms. The subformula $\tau(r = \overline{1}..\overline{10})$ is $\top$ if r is one of the numerals $\overline{1}, \ldots, \overline{10}$, and $\bot$ otherwise.

The semantics of MGC described in this section, like the semantics of aggregates proposed by Gebser $et\ al.$ (2015), aims at modeling the behavior of the answer set solver CLINGO. The definition in this paper is simpler than the 2015 version, but more limited in scope, because it is does not cover aggregates other than #count. The two versions of the semantics of #count are not completely equivalent, however, because they handle infinite sets in slightly different ways. This difference does not affect safe programs and is in this sense inessential.

6 Review: Representing mini-gringo rules by formulas

The target language of the translation τ^* is a first-order language with two sorts: the sort $general$ and its subsort $integer$. General variables are meant to range over arbitrary precomputed terms, and we identify them with variables used in mini-GRINGO rules. Integer of the second sort is meant to range over numerals (or, equivalently, integers). The signature σ_0 of the language includes

- all precomputed terms as object constants; an object constant is assigned the sort $integer$ iff it is a numeral;
- the symbols $+$, $-$ and $\times$ as binary function constants; their arguments and values have the sort $integer$;
- symbols p/n, where p is a symbolic constant, as n-ary predicate constants;
- comparison symbols (6) as binary predicate constants.

An atomic formula $(p/n)(\mathbf{t})$ can be abbreviated as $p(\mathbf{t})$. An atomic formula $\prec(t_1, t_2)$, where $\prec$ is a comparison symbol, can be written as $t_1 \prec t_2$.

Lifschitz $et\ al.$ (2019) defined, for every mini-GRINGO term t, a formula $val_t(Z)$ that expresses, informally speaking, that Z is one of the values of t. For example, $val_{\overline{2}}(Z)$ is $Z = \overline{2}$. If $\mathbf{t}$ is a tuple $t_1, \ldots, t_n$ of mini-GRINGO terms, and $\mathbf{Z}$ is a tuple $Z_1, \ldots, Z_n$ of distinct general variables, then $val_{\mathbf{t}}(\mathbf{Z})$ stands for $val_{t_1}(Z_1) \wedge \cdots \wedge val_{t_n}(Z_n)$.

The translation τ^B, which transforms literals and comparisons into formulas over the signature σ_0, is defined in that paper as follows:[4]

- $\tau^B(p(\mathbf{t})) = \exists \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \wedge p(\mathbf{Z}))$;
- $\tau^B(not\ p(\mathbf{t})) = \exists \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \wedge \neg p(\mathbf{Z}))$;

---

[4] The superscript B indicates that this translation is intended for $bodies$ of rules.

- $\tau^B(not\ not\ p(\mathbf{t})) = \exists \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \wedge \neg\neg p(\mathbf{Z}));$
- $\tau^B(t_1 \prec t_2) = \exists Z_1 Z_2(val_{t_1}(Z_1) \wedge val_{t_2}(Z_2) \wedge Z_1 \prec Z_2).$

Here Z_1, Z_2, and members of the tuple $\mathbf{Z}$ are fresh general variables.

The result of applying τ^* to a mini-GRINGO rule $H \leftarrow B_1 \wedge \cdots \wedge B_n$ can be defined as the universal closure of the formula

$$
\begin{array}{ll}
B_1^* \wedge \cdots \wedge B_n^* \wedge val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z}) & \text{if } H \text{ is } p(\mathbf{t}), \\
B_1^* \wedge \cdots \wedge B_n^* \wedge val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z}) \vee \neg p(\mathbf{Z}) & \text{if } H \text{ is } p\{(\mathbf{t})\}, \\
\neg(B_1^* \wedge \cdots \wedge B_n^*) & \text{if } H \text{ is empty,}
\end{array}
\tag{18}
$$

where B_i^* stands for $\tau^B(B_i)$, and $\mathbf{Z}$ is a tuple of fresh general variables.

For example, the result of applying τ^* to choice rule (11) is

$$
\forall XZ(\tau^B(p(X)) \wedge Z = \overline{2} \to p(Z) \vee \neg p(Z));
$$

$\tau^B(p(X))$ can be further expanded into $\exists Z(Z = X \wedge p(Z))$.

7 Extending the target language

In this section, we extend the translation τ^* to arbitrary pure rules. This more general translation produces first-order formulas over the signature σ_1 that is obtained from σ_0 by adding infinitely many predicate constants

$$
Atleast_F^{\mathbf{X};\mathbf{V}} \text{ and } Atmost_F^{\mathbf{X};\mathbf{V}},
\tag{19}
$$

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 $\mathbf{V}$.[5] The number of arguments of each of constants (19) 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 $\mathbf{X}$; symbolically,

$$
\exists \mathbf{X}_1 \cdots \mathbf{X}_n \left(\bigwedge_{i=1}^{n} F_{\mathbf{X}_i}^{\mathbf{X}} \wedge \bigwedge_{i<j} \neg(\mathbf{X}_i = \mathbf{X}_j) \right),
\tag{20}
$$

where $\mathbf{X}_1, \ldots, \mathbf{X}_n$ are tuples of fresh general variables.[6] For any precomputed term r, the expression $\exists_{\geq r} \mathbf{X} F$ will stand for

$$
\begin{array}{ll}
\text{formula (20),} & \text{if } r = \overline{n} > \overline{0}, \\
\top, & \text{if } r \leq \overline{0}, \\
\bot, & \text{if } r > \overline{n} \text{ for all integers } n.
\end{array}
$$

The formula $Atmost_F^{\mathbf{X},\mathbf{V}}(\mathbf{V}, \overline{n})$ is meant to express that F holds for at most n values of $\mathbf{X}$; symbolically,

$$
\forall \mathbf{X}_1 \cdots \mathbf{X}_{n+1} \left(\bigwedge_{i=1}^{n+1} F_{\mathbf{X}_i}^{\mathbf{X}} \to \bigvee_{i<j} \mathbf{X}_i = \mathbf{X}_j \right).
\tag{21}
$$

---

[5] Adding infinitely many predicate symbols gives us a single signature that is sufficient for representing all MGC rules. The translation of any specific rule will only contain finitely many symbols, of course.

[6] An expression of the form $(X_1, X_2, \ldots) = (Y_1, Y_2, \ldots)$ stands for $X_1 = Y_1 \wedge X_2 = Y_2 \wedge \cdots$.

For any precomputed term r, the expression $\exists_{\leq r}\mathbf{X}F$ will stand for

$$
\begin{array}{ll}
\text{formula (21),} & \text{if } r = \overline{n} \geq \overline{0}, \\
\bot, & \text{if } r < \overline{0}, \\
\top, & \text{if } r > \overline{n} \text{ for all integers } n.
\end{array}
$$

The set of all sentences of the forms

$$
\forall \mathbf{V} \left(Atleast_F^{\mathbf{X};\mathbf{V}}(\mathbf{V},r) \leftrightarrow \exists_{\geq r}\mathbf{X}F \right), \tag{22}
$$

$$
\forall \mathbf{V} \left(Atmost_F^{\mathbf{X};\mathbf{V}}(\mathbf{V},r) \leftrightarrow \exists_{\leq r}\mathbf{X}F \right) \tag{23}
$$

will be denoted by *Defs*.

8 Representing pure rules by formulas

To extend the definition of τ^* reproduced in Section 6 to arbitrary pure rules, we need to say how to choose B_i^* in (18) when B_i includes an aggregate element $\mathbf{X} : \mathbf{L}$. Let $\mathbf{V}$ be the list of global variables that occur in $\mathbf{L}$, and let $\mathbf{W}$ be the list of local variables that occur in $\mathbf{L}$ but are not included in $\mathbf{X}$. Then, B_i^* is defined as

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

if B_i is $count\{\mathbf{X} : \mathbf{L}\} \geq t$, and as

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

if B_i is $count\{\mathbf{X} : \mathbf{L}\} \leq t$, where C is a fresh general variable.

For example, the result of applying τ^* to rule (13) is

$$
\exists C \left(val_{\overline{2}}(C) \wedge Atmost_{\exists Y \tau^B(p(X,Y))}^{X;}(C) \right) \rightarrow q,
$$

($\mathbf{V}$ is empty, $\mathbf{W}$ is Y). The result of applying τ^* to rule (14) is

$$
\forall Y \left(\exists C \left(val_{\overline{2}}(C) \wedge Atmost_{\tau^B(p(X,Y))}^{X;Y}(Y,C) \right) \wedge \tau^B(Y = \overline{1} .. \overline{10}) \rightarrow q \right),
$$

($\mathbf{V}$ is Y, $\mathbf{W}$ is empty).

For any MGC program Π, $\tau^*(\Pi)$ stands for the conjunction of the formulas $\tau^*(R)$ for all rules R of Π. Thus, $\tau^*(\Pi)$ is a sentence over the signature σ_1.

9 Review: infinitary logic of here-and-there

Some of the properties of the translation τ^* discussed below refer to the deductive system of infinitary propositional logic of here-and-there (Harrison *et al.* 2017), denoted by HT^∞. In this section, we reproduce the definition of that system.

The derivable objects of HT^∞ are *sequents*—expressions of the form $\Gamma \Rightarrow F$, where F is an infinitary propositional formula, and Γ is a finite set of infinitary propositional formulas ("F under assumptions Γ"). To simplify notation, we write Γ as a list. We identify a sequent of the form $\Rightarrow F$ with the formula F.

Table 1. *Introduction and elimination rules of infinitary propositional logic. By $\mathcal{H}^\wedge$ and $\mathcal{H}^\vee$, we denote the conjunction and disjunction of all formulas in $\mathcal{H}$.*

$$(\wedge I)\ \frac{\Gamma \Rightarrow H \quad \text{for all } H \in \mathcal{H}}{\Gamma \Rightarrow \mathcal{H}^\wedge} \qquad\qquad (\wedge E)\ \frac{\Gamma \Rightarrow \mathcal{H}^\wedge}{\Gamma \Rightarrow H}\quad (H \in \mathcal{H})$$

$$(\vee I)\ \frac{\Gamma \Rightarrow H}{\Gamma \Rightarrow \mathcal{H}^\vee}\quad (H \in \mathcal{H}) \qquad\qquad (\vee E)\ \frac{\Gamma \Rightarrow \mathcal{H}^\vee \qquad \Delta, H \Rightarrow F \quad \text{for all } H \in \mathcal{H}}{\Gamma, \Delta \Rightarrow F}$$

$$(\to I)\ \frac{\Gamma, F \Rightarrow G}{\Gamma \Rightarrow F \to G} \qquad\qquad (\to E)\ \frac{\Gamma \Rightarrow F \quad \Delta \Rightarrow F \to G}{\Gamma, \Delta \Rightarrow G}$$

The axiom schemas of HT^∞ are

$$F \Rightarrow F,$$

$$F \vee (F \to G) \vee \neg G,$$

and

$$\bigwedge_{\alpha \in A} \bigvee_{F \in \mathcal{H}_\alpha} F \to \bigvee_{(F_\alpha)_{\alpha \in A}} \bigwedge_{\alpha \in A} F_\alpha, \tag{24}$$

where $(\mathcal{H}_\alpha)_{\alpha \in A}$ is a nonempty family of sets of formulas; the disjunction in the consequent of (24) extends over all elements $(F_\alpha)_{\alpha \in A}$ of the Cartesian product of the family $(\mathcal{H}_\alpha)_{\alpha \in A}$. The inference rules of HT^∞ are the introduction and elimination rules for the propositional connectives shown in the Table 1 above and the weakening rule

$$(W)\ \frac{\Gamma \Rightarrow F}{\Gamma, \Delta \Rightarrow F}.$$

Falsity and negation are not mentioned in the axiom schemas and inference rules of HT^∞ because $\bot$ is considered shorthand for $\emptyset^\vee$, and $\neg F$ is shorthand for $F \to \bot$.

The set of *theorems of HT^∞* is the smallest set of sequents that includes the axioms of the system and is closed under the application of its inference rules. We say that formulas F and G are *equivalent in HT^∞* if $F \leftrightarrow G$ is a theorem of HT^∞.

The role of this deductive system is determined by the fact that two infinitary propositional formulas are strongly equivalent to each other if and only if they are equivalent in HT^∞ (Harrison *et al.* 2017, Corollary 2).

10 Properties of the generalized translation

Informally speaking, a pure rule R has the same meaning as the sentence $\tau^*(R)$. This claim is made precise in Theorem 1 below. The statement of the theorem refers to the infinitary propositional formulas obtained from sentences over σ_1 by applying the grounding operator gr, which is defined recursively:

- $gr(\bot)$ is $\bot$;
- if F is $\prec (t_1, t_2)$, where $\prec$ is a comparison symbol, then $gr(F)$ is $\top$ if the relation $\prec$ holds for the values of t_1 and t_2, and $\bot$ otherwise;
- if F is $p(\mathbf{t})$, where p is not a comparison symbol, then $gr(F)$ is obtained from F by replacing each member of the tuple $\mathbf{t}$ by its value;

- $gr(F \odot G)$ is $gr(F) \odot gr(G)$ for every binary connective $\odot$;
- $gr(\forall X\, F)$ is the conjunction of the formulas $gr\left(F_r^X\right)$ over all precomputed terms r if X is a general variable, and over all numerals r if X is an integer variable;
- $gr(\exists X\, F)$ is the disjunction of the formulas $gr\left(F_r^X\right)$ over all precomputed terms r if X is a general variable, and over all numerals r if X is an integer variable.

Thus, $gr(F)$ is an infinitary propositional formula over the signature consisting of all atomic formulas of the form $p(\mathbf{r})$, where p is different from comparison symbols and $\mathbf{r}$ is a tuple of precomputed terms. Such atomic formulas will be called *extended precomputed atoms*. Unlike precomputed atoms, they may contain predicate symbols (19). If Γ is a set of sentences over σ_1, then $gr(\Gamma)$ stands for the set of formulas $gr(F)$ for all F in Γ.

The statement of the theorem refers also to the system HT^∞ (Section 9) extended by the axioms $gr(Defs)$. These axioms express the meaning of predicate symbols (19).

Theorem 1
For any pure rule R, $gr(\tau^*(R))$ is equivalent to $\tau(R)$ in $HT^\infty + gr(Defs)$.

About MGC programs Π_1, Π_2 we say that they are *strongly equivalent* to each other if $\tau(\Pi_1)$ is strongly equivalent to $\tau(\Pi_2)$. This condition guarantees that for any MGC program Π (and, more generally, for any logic program Π in a similar language), $\Pi_1 \cup \Pi$ has the same stable models as $\Pi_2 \cup \Pi$.

Theorem 2
MGC programs Π_1, Π_2 are strongly equivalent to each other iff

$$gr(\tau^*(\Pi_1)) \text{ is equivalent to } gr(\tau^*(\Pi_2)) \text{ in } HT^\infty + gr(Defs). \tag{25}$$

Thus, the claim that MGC programs Π_1, Π_2 are strongly equivalent to each other can be always established, in principle, by deriving each of the infinitary propositional formulas $gr(\tau^*(\Pi_1))$, $gr(\tau^*(\Pi_2))$ from the other in $HT^\infty + gr(Defs)$. Theorem 3 below shows that in some cases such a claim can be justified by operating with finite formulas—with first-order formulas of the signature σ_1. Instead of $HT^\infty + gr(Defs)$, we can use the logic of here-and-there with arithmetic (Lifschitz 2021) extended by the axiom schemas *Defs*:

Theorem 3
For any MGC programs Π_1, Π_2, if the formulas $\tau^*(\Pi_1)$ and $\tau^*(\Pi_2)$ are equivalent in $HTA + Defs$ then Π_1 and Π_2 are strongly equivalent to each other.

As an example, we will use $HT^\infty + gr(Defs)$ to verify that rules (1), (2), (3) are strongly equivalent to each other. The translation τ^* transforms these rules into the formulas

$$\forall Y \left(\exists C \left(C = Y \land Atleast^{X;}_{\tau^B(p(X))}(C) \right) \land \tau^B(Y = \overline{1}), \to q \right), \tag{26}$$

$$\exists C \left(C = \overline{1} \land Atleast^{X;}_{\tau^B(p(X))}(C) \right) \to q, \tag{27}$$

$$\forall X (\tau^B(p(X)) \to q). \tag{28}$$

The first two formulas are equivalent to each other in intuitionistic predicate calculus with equality, which is a subsystem of HTA; this is clear from the fact that $\tau^B(Y = \overline{1})$

stands for the formula $\exists Z_1 Z_2 (Z_1 = Y \wedge Z_2 = \overline{1} \wedge Z_1 = Z_2)$, which is intuitionistically equivalent to $Y = \overline{1}$. Furthermore, (27) is intuitionistically equivalent to

$$Atleast^{X;}_{\tau^B(p(X))}(\overline{1}) \to q. \tag{29}$$

Using the axiom

$$Atleast^{X;}_{\tau^B(p(X))}(\overline{1}) \leftrightarrow \exists X \tau^B(p(X)),$$

of $HTA + Defs$, (29) can be transformed into the formula $\exists X \, \tau^B(p(X)) \to q$, which is intuitionistically equivalent to (28).

11 Proofs

In this section, the word "equivalent" in application to infinitary propositional formulas refers to equivalence in HT^∞ whenever the deductive system is not specified.

Lemma 1
For any tuple $\mathbf{t}$ of terms in the language of mini-GRINGO and any tuple $\mathbf{r}$ of precomputed terms of the same length, the formula $gr(val_\mathbf{t}(\mathbf{r}))$ is provable in HT^∞ if $\mathbf{r} \in [\mathbf{t}]$, and refutable otherwise.

Proof
For the case when $\mathbf{t}$ is a single term, the assertion of the lemma can be proved by induction (Lifschitz *et al.* 2019, Proposition 1). The general case easily follows. □

The following fact is Proposition 2 by Lifschitz *et al.* (2019).

Lemma 2
If L is a ground literal or ground comparison in the language of mini-GRINGO then $gr(\tau^B(L))$ is equivalent to $\tau(L)$.

Lemma 3
Let $\mathbf{X}$, $\mathbf{V}$, $\mathbf{W}$ be disjoint lists of distinct general variables, and let A be an aggregate atom $count\{\mathbf{X} : \mathbf{L}\} \prec t$ such that every variable occurring in $\mathbf{L}$ belongs to one of these three lists, and every variable occurring in t belongs to $\mathbf{V}$. For any list $\mathbf{v}$ of precomputed terms of the same length as $\mathbf{V}$, the formula $\tau \left(A_\mathbf{v}^\mathbf{V} \right)$ is equivalent in $HT^\infty + gr(Defs)$ to

$$gr \left(\exists C \left(val_{t_\mathbf{v}^\mathbf{V}}(C) \wedge Atleast^{\mathbf{X};\mathbf{V}}_{\exists \mathbf{W} \tau^B(\mathbf{L})}(\mathbf{v}, C) \right) \right), \tag{30}$$

if $\prec$ is $\geq$, and to

$$gr \left(\exists C \left(val_{t_\mathbf{v}^\mathbf{V}}(C) \wedge Atmost^{\mathbf{X};\mathbf{V}}_{\exists \mathbf{W} \tau^B(\mathbf{L})}(\mathbf{v}, C) \right) \right), \tag{31}$$

if $\prec$ is $\leq$.

Proof
Case 1: $\prec$ is $\geq$. Formula (30) can be written as

$$\bigvee_c \left(gr(val_{t_\mathbf{v}^\mathbf{V}}(c)) \wedge Atleast^{\mathbf{X};\mathbf{V}}_{\exists \mathbf{W} \tau^B(\mathbf{L})}(\mathbf{v}, c) \right),$$

where c ranges over precomputed terms. From Lemma 1, we see that it is equivalent to

$$\bigvee_{c \in [t_{\mathbf{v}}^{\mathbf{V}}]} Atleast_{\exists \mathbf{W} \tau^B(\mathbf{L})}^{\mathbf{X};\mathbf{V}}(\mathbf{v}, c).$$

Consider the infinitary formula obtained by grounding (22) with $\exists \mathbf{W} \tau^B(\mathbf{L})$ as F. One of its conjunctive terms is

$$Atleast_{\exists \mathbf{W} \tau^B(\mathbf{L})}^{\mathbf{X};\mathbf{V}}(\mathbf{v}, c) \leftrightarrow gr\left(\left(\exists_{\geq c}\mathbf{X}\exists\mathbf{W}\tau^B(\mathbf{L})\right)_{\mathbf{v}}^{\mathbf{V}}\right).$$

Consequently (30) is equivalent in $HT^\infty + gr(Defs)$ to

$$\bigvee_{c \in [t_{\mathbf{v}}^{\mathbf{V}}]} gr\left(\left(\exists_{\geq c}\mathbf{X}\exists\mathbf{W}\tau^B(\mathbf{L})\right)_{\mathbf{v}}^{\mathbf{V}}\right). \tag{32}$$

Case 1.1: The set $[t_{\mathbf{v}}^{\mathbf{V}}]$ is empty. Then (32) is the empty disjunction $\bot$; $\tau\left(A_{\mathbf{v}}^{\mathbf{V}}\right)$ is $\bot$ as well. *Case 1.2:* The set $[t_{\mathbf{v}}^{\mathbf{V}}]$ is nonempty. Since t is interval-free, this set is a singleton $\{c\}$, so that (32) is

$$gr\left(\left(\exists_{\geq c}\mathbf{X}\exists\mathbf{W}\tau^B(\mathbf{L})\right)_{\mathbf{v}}^{\mathbf{V}}\right), \tag{33}$$

and $\tau\left(A_{\mathbf{v}}^{\mathbf{V}}\right)$ is

$$\bigvee_{\Delta \,:\, \overline{|\Delta|} \geq c} \bigwedge_{\mathbf{x} \in \Delta} \bigvee_{\mathbf{w}} \tau\left(\mathbf{L}_{\mathbf{x},\,\mathbf{v},\,\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right). \tag{34}$$

Case 1.2.1: $c \leq \overline{0}$. Then (33) is $\top$. The disjunctive term of (34) with $\Delta = \emptyset$ is the empty conjunction $\top$, so that (34) is equivalent to $\top$. *Case 1.2.2:* for all n, $c > \overline{n}$. Then (33) is $\bot$. Formula (34) is the empty disjunction $\bot$ as well. *Case 1.2.3:* c is a numeral $\overline{n}$, $n > 0$. Then (33) is

$$gr\left(\exists\mathbf{X}_1 \cdots \mathbf{X}_n\left(\bigwedge_{i=1}^{n}\exists\mathbf{W}\left(\tau^B(\mathbf{L})_{\mathbf{X}_i,\mathbf{v}}^{\mathbf{X},\mathbf{V}}\right) \wedge \bigwedge_{i<j}\neg(\mathbf{X}_i = \mathbf{X}_j)\right)\right),$$

This formula can be rewritten as

$$\bigvee_{\mathbf{x}_1,\ldots,\mathbf{x}_n}\left(\bigwedge_{i=1}^{n}\bigvee_{\mathbf{w}} gr\left((\tau^B\mathbf{L})_{\mathbf{x}_i,\mathbf{v},\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right) \wedge \bigwedge_{i<j}\neg gr(\mathbf{x}_i = \mathbf{x}_j)\right)$$

$(\mathbf{x}_1,\ldots\mathbf{x}_n,\mathbf{w}$ range over tuples of precomputed terms). The part $\bigwedge_{i<j}\neg(gr(\mathbf{x}_i = \mathbf{x}_j))$ is equivalent to $\top$ if the tuples $\mathbf{x}_1,\ldots\mathbf{x}_n$ are pairwise distinct, that is to say, if the cardinality of the set $\{\mathbf{x}_1,\ldots\mathbf{x}_n\}$ is n; otherwise, this conjunction is equivalent to $\bot$. Consequently (33) is equivalent to

$$\bigvee_{\Delta \,:\, |\Delta| = n} \bigwedge_{\mathbf{x} \in \Delta} \bigvee_{\mathbf{w}} gr\left((\tau^B\mathbf{L})_{\mathbf{x},\,\mathbf{v},\,\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right).$$

The formula $gr\left((\tau^B\mathbf{L})_{\mathbf{x},\,\mathbf{v},\,\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right)$ can be rewritten as $gr\left(\tau^B\left(\mathbf{L}_{\mathbf{x},\,\mathbf{v},\,\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right)\right)$. By Lemma 2, it is equivalent to $\tau\left(\mathbf{L}_{\mathbf{x},\,\mathbf{v},\,\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right)$, so that (33) is equivalent to

$$\bigvee_{\Delta \,:\, |\Delta| = n} \bigwedge_{\mathbf{x} \in \Delta} \bigvee_{\mathbf{w}} \tau\left(\mathbf{L}_{\mathbf{x},\,\mathbf{v},\,\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right). \tag{35}$$

Disjunction (34) can be obtained from this disjunction by adding similar disjunctive terms with sets Δ containing more than n tuples. Since each of these disjunctive terms is stronger than some of the disjunctive terms in (35), the two disjunctions are equivalent.

Case 2: $\prec$ is $\leq$. Formula (31) is equivalent in $HT^\infty + gr(Defs)$ to

$$\bigvee_{c \in [t_{\mathbf{v}}^{\mathbf{V}}]} gr\left(\left(\exists_{\leq c} \mathbf{X} \exists \mathbf{W} \tau^B(\mathbf{L})\right)_{\mathbf{v}}^{\mathbf{V}}\right); \tag{36}$$

this is parallel to the argument in Case 1.

Case 2.1: The set $\left[t_{\mathbf{v}}^{\mathbf{V}}\right]$ is empty. Then (36) is the empty disjunction $\bot$; $\tau\left(A_{\mathbf{v}}^{\mathbf{V}}\right)$ is $\bot$ as well. *Case 2.2:* The set $\left[t_{\mathbf{v}}^{\mathbf{V}}\right]$ is nonempty. Since t is interval-free, this set is a singleton $\{c\}$, so that (36) is

$$gr\left(\left(\exists_{\leq c} \mathbf{X} \exists \mathbf{W} \tau^B(\mathbf{L})\right)_{\mathbf{v}}^{\mathbf{V}}\right), \tag{37}$$

and $\tau\left(A_{\mathbf{v}}^{\mathbf{V}}\right)$ is

$$\bigwedge_{\Delta \,:\, \overline{|\Delta|} > c} \neg \bigwedge_{\mathbf{x} \in \Delta} \bigvee_{\mathbf{w}} \tau(\mathbf{L}_{\mathbf{x},\,\mathbf{v},\,\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}). \tag{38}$$

Case 2.2.1: $c < \overline{0}$. Then (37) is $\bot$. The conjunctive term of (38) with $\Delta = \emptyset$ is $\neg\top$, so that (38) is equivalent to $\bot$. *Case 2.2.2:* for all n, $c > \overline{n}$. Then (37) is $\top$. Formula (38) is the empty conjunction $\top$ as well. *Case 2.2.3:* c is a numeral $\overline{n}$, $n > 0$. Then (37) is

$$gr\left(\forall \mathbf{X}_1 \cdots \mathbf{X}_{n+1} \left(\bigwedge_{i=1}^{n+1} \exists \mathbf{W}\left(\tau^B(\mathbf{L})_{\mathbf{X}_i,\mathbf{v}}^{\mathbf{X},\mathbf{V}}\right) \to \bigvee_{i<j} \mathbf{X}_i = \mathbf{X}_j\right)\right).$$

This formula can be rewritten as

$$\bigwedge_{\mathbf{x}_1,\ldots,\mathbf{x}_{n+1}} \left(\bigwedge_{i=1}^{n+1} \bigvee_{\mathbf{w}} gr\left((\tau^B \mathbf{L})_{\mathbf{x}_i,\mathbf{v},\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right) \to \bigvee_{i<j} gr(\mathbf{x}_i = \mathbf{x}_j)\right).$$

The consequent $\bigvee_{i<j} gr(\mathbf{x}_i = \mathbf{x}_j))$ is equivalent to $\bot$ if the tuples $\mathbf{x}_1, \ldots \mathbf{x}_{n+1}$ are pairwise distinct, that is to say, if the cardinality of the set $\{\mathbf{x}_1, \ldots \mathbf{x}_{n+1}\}$ is $n+1$; otherwise this conjunction is equivalent to $\top$. Consequently, (37) is equivalent to

$$\bigwedge_{\Delta \,:\, |\Delta| = n+1} \neg \bigwedge_{\mathbf{x} \in \Delta} \bigvee_{\mathbf{w}} gr\left((\tau^B \mathbf{L})_{\mathbf{x},\,\mathbf{v},\,\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right),$$

and furthermore to

$$\bigwedge_{\Delta \,:\, |\Delta| = n+1} \neg \bigwedge_{\mathbf{x} \in \Delta} \bigvee_{\mathbf{w}} \tau\left(\mathbf{L}_{\mathbf{x},\,\mathbf{v},\,\mathbf{w}}^{\mathbf{X},\mathbf{V},\mathbf{W}}\right); \tag{39}$$

this is parallel to the argument in Case 1.2.3. Conjunction (38) can be obtained from this conjunction by adding similar conjunctive terms with finite sets Δ containing more than $n+1$ tuples. Since each of these conjunctive terms is weaker than some of the conjunctive terms in (39), the two conjunctions are equivalent to each other. $\square$

Proof of Theorem 1

Assume, for instance, that R is a basic rule $p(\mathbf{t}) \leftarrow B_1 \wedge \cdots \wedge B_n$; for choice rules and constraints the proof is similar. Then $\tau^*(R)$ is

$$\forall \mathbf{V}\mathbf{Z}(B_1^* \wedge \cdots \wedge B_n^* \wedge val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z})),$$

where $\mathbf{V}$ is the list of global variables of R, and B_i^* are the formulas defined in Section 8. It follows that $gr(\tau^*(R))$ is the conjunction of the formulas

$$gr\left((B_1^*)_{\mathbf{v}}^{\mathbf{V}}\right) \wedge \cdots \wedge gr\left((B_n^*)_{\mathbf{v}}^{\mathbf{V}}\right) \wedge gr\left(val_{\mathbf{t}_{\mathbf{v}}^{\mathbf{V}}}(\mathbf{r})\right) \to p(\mathbf{r}),$$

over all tuples $\mathbf{v}$ of precomputed terms of the same length as $\mathbf{V}$ and all tuples $\mathbf{r}$ of precomputed terms of the same length as $\mathbf{Z}$. By Lemma 1, we can conclude that $gr(\tau^*(R))$ is equivalent to the formula

$$\bigwedge_{\mathbf{v}} \left(\left(gr\left((B_1^*)_{\mathbf{v}}^{\mathbf{V}}\right) \wedge \cdots \wedge gr\left((B_n^*)_{\mathbf{v}}^{\mathbf{V}}\right)\right) \to \bigwedge_{\mathbf{r}\in[\mathbf{t}_{\mathbf{v}}^{\mathbf{V}}]} p(\mathbf{r}) \right).$$

Each of the formulas

$$gr\left((B_i^*)_{\mathbf{v}}^{\mathbf{V}}\right), \tag{40}$$

$(i = 1, \ldots, n)$ is equivalent in $HT^\infty + gr(Defs)$ to $\tau\left((B_i)_{\mathbf{v}}^{\mathbf{V}}\right)$. Indeed, if B_i is a literal or a comparison, then (40) is $gr\left(\left(\tau^B(B_i)\right)_{\mathbf{v}}^{\mathbf{V}}\right)$, which can be also written as $gr\left(\tau^B\left((B_i)_{\mathbf{v}}^{\mathbf{V}}\right)\right)$; this formula is equivalent in $HT^\infty + gr(Defs)$ to $\tau\left((B_i)_{\mathbf{v}}^{\mathbf{V}}\right)$ by Lemma 2. If B_i is an aggregate atom then (40) is equivalent in $HT^\infty + gr(Defs)$ to $\tau\left((B_i)_{\mathbf{v}}^{\mathbf{V}}\right)$ by Lemma 3. Consequently $gr(\tau^*(R))$ is equivalent in $HT^\infty + gr(Defs)$ to

$$\bigwedge_{\mathbf{v}} \left(\tau\left((B_1)_{\mathbf{v}}^{\mathbf{V}}\right) \wedge \cdots \wedge \tau\left((B_n)_{\mathbf{v}}^{\mathbf{V}}\right) \to \bigwedge_{\mathbf{r}\in[\mathbf{t}_{\mathbf{v}}^{\mathbf{V}}]} p(\mathbf{r}) \right). \tag{41}$$

It remains to observe that instances of R are rules of the form

$$p(\mathbf{t}_{\mathbf{v}}^{\mathbf{V}}) \leftarrow (B_1)_{\mathbf{v}}^{\mathbf{V}} \wedge \cdots \wedge (B_n)_{\mathbf{v}}^{\mathbf{V}},$$

so that (41) is $\tau(R)$.

Lemma 4

If an infinitary propositional formula over the set of precomputed atoms is provable in $HT^\infty + gr(Defs)$, then it is provable in HT^∞.

Proof (sketch)

The set $gr(Defs)$ consists of infinitary propositional formulas of the forms

$$\bigwedge_{\mathbf{v}} \left(Atleast_F^{\mathbf{X};\mathbf{V}}(\mathbf{v}, r) \leftrightarrow gr\left(\exists_{\geq r}\mathbf{X}F_{\mathbf{v}}^{\mathbf{V}}\right) \right), \tag{42}$$

and

$$\bigwedge_{\mathbf{v}} \left(Atmost_F^{\mathbf{X};\mathbf{V}}(\mathbf{v}, r) \leftrightarrow gr\left(\exists_{\leq r}\mathbf{X}F_{\mathbf{v}}^{\mathbf{V}}\right) \right). \tag{43}$$

A derivation from $gr(Defs)$ can be visualized as a tree with axioms of HT^∞ and formulas (42), (43), attached to leaves. In such a tree, modify all formulas by replacing

- atoms $Atleast_F^{\mathbf{X};\mathbf{V}}(\mathbf{v}, r)$ by $gr\left(\exists_{\leq r}\mathbf{X}F_{\mathbf{v}}^{\mathbf{V}}\right)$, and
- atoms $Atmost_F^{\mathbf{X};\mathbf{V}}(\mathbf{v}, r)$ by $gr\left(\exists_{\geq r}\mathbf{X}F_{\mathbf{v}}^{\mathbf{V}}\right)$.

The result is a derivation from formulas that are provable in HT^∞. If the formula attached to the root does not contain $Atleast_F^{\mathbf{X};\mathbf{V}}$, $Atmost_F^{\mathbf{X};\mathbf{V}}$, then it is not affected by this transformation.

Proof of Theorem 2
By Theorem 1, each of the equivalences

$$gr(\tau^*(\Pi_1)) \leftrightarrow \tau(\Pi_1), \ gr(\tau^*(\Pi_2)) \leftrightarrow \tau(\Pi_2),$$

is provable in $HT^\infty + gr(Defs)$. Consequently condition (25) is equivalent to the condition

$$\tau(\Pi_1) \text{ is equivalent to } \tau(\Pi_2) \text{ in } HT^\infty + gr(Defs). \tag{44}$$

By Lemma 4, (44) is equivalent to the condition

$$\tau(\Pi_1) \leftrightarrow \tau(\Pi_2) \text{ is provable in } HT^\infty,$$

which holds if and only if Π_1 is strongly equivalent to Π_2.

Lemma 5
If a sentence F over the signature σ_1 is provable in $HTA + Defs$, then $gr(F)$ is provable in $HT^\infty + gr(Defs)$.

Proof (sketch)
For any axiom S of HTA, the formula $gr(S)$ is provable in HT^∞. (To be precise, axioms of HTA are sequents, and the transformation gr needs to be applied to the universal closure of the formula corresponding to S.) For any instance

$$\frac{S_1 \ \cdots \ S_k}{S},$$

of an inference rule of HTA, the formula $gr(S)$ is derivable from $gr(S_1), \ldots, gr(S_k)$ in HT^∞. It follows that for any formula F that is derivable from $Defs$ in HTA, the formula $gr(F)$ is derivable from $gr(Defs)$ in HT^∞.

Proof of Theorem 3
By Lemma 5, if the equivalence $\tau^*(\Pi_1) \leftrightarrow \tau^*(\Pi_2)$ is provable in $HTA + Defs$ then the equivalence $gr(\tau^*(\Pi_1)) \leftrightarrow gr(\tau^*(\Pi_2))$ is provable in $HT^\infty + gr(Defs)$. Then, by Theorem 2, the programs Π_1 and Π_2 are strongly equivalent.

12 Related Work

Fandinno *et al.* (2022) defined a translation similar to τ^* for an answer set programming language that is in some ways less expressive than mini-GRINGO with counting (no arithmetic operations), and in some ways more expressive (the `#sum` aggregate is allowed, besides `#count`). The main difference between that approach to transforming aggregate expressions into formulas and the one described above is that the former employs function symbols in the role that predicate symbols (19) play here. As discussed in Footnote 1, thinking of `#count` as a function may be misleading. This is apparently the reason why the adequacy of the translation due to Fandinno *et al.* is only guaranteed for programs without positive recursion through aggregates (Fandinno *et al.* 2022, Theorem 3). This assumption is not satisfied, for instance, for program (4), (5).

The technical problems discussed in this paper are specific for the approach to aggregates implemented in the answer set solver CLINGO and do not appear in the same form, for instance, in the theory of the solver DLV (Faber *et al.* 2011). The semantics of aggregates based on the vicious circle principle (Gelfond and Zhang 2019), unlike the CLINGO semantics, makes rule (4) strongly equivalent to each of the rules (1)–(3).

Acknowledgments

References

FABER, W., PFEIFER, G. AND LEONE, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence 175*, 278–298.

FANDINNO, J., NANSEN, Z. AND LIERLER, Y. 2022. Axiomatization of aggregates in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*. To appear.

GEBSER, M., HARRISON, A., KAMINSKI, R., LIFSCHITZ, V. AND SCHAUB, T. 2015. Abstract Gringo. *Theory and Practice of Logic Programming 15*, 449–463.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., LINDAUER, M., OSTROWSKI, M., ROMERO, J., SCHAUB, T. AND THIELE, S. 2019. Potassco User Guide. URL: `https://github.com/potassco/guide/releases/`

GELFOND, M. AND KAHL, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press.

GELFOND, M. AND ZHANG, Y. 2019. Vicious circle principle, aggregates, and formation of sets in ASP based languages. *Artificial Intelligence 275*, 28–77.

HARRISON, A., LIFSCHITZ, V., PEARCE, D. AND VALVERDE, A. 2017. Infinitary equilibrium logic and strongly equivalent logic programs. *Artificial Intelligence 246*, 22–33.

LIFSCHITZ, V. 2019. *Answer Set Programming*. Springer.

LIFSCHITZ, V. 2021. Here and there with arithmetic. *Theory and Practice of Logic Programming*.

LIFSCHITZ, V., LÜHNE, P. AND SCHAUB, T. 2019. 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*.

LIFSCHITZ, V., PEARCE, D. AND VALVERDE, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2*, 526–541.

MAREK, V. AND TRUSZCZYNSKI, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*. Springer Verlag, 375–398.

NIEMELÄ, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25*, 241–273.

TRUSZCZYNSKI, M. 2012. Connecting first-order ASP and the logic FO(ID) through reducts. In *Correct Reasoning: Essays on Logic-Based AI in Honor of Vladimir Lifschitz*, E. Erdem, J. Lee, Y. Lierler and D. Pearce, Eds. Springer, 543–559.