Generalizing the Syntax of Terms in Mini-GRINGO

Vladimir Lifschitz

University of Texas at Austin

Abstract. In answer set programming, a program verification task can be sometimes accomplished by transforming rules into first-order sentences so that the task is reduced to reasoning in classical logic, and invoking a resolution theorem prover. The proof assistant ANTHEM, which implements this idea, allows us to reason about programs written in mini-GRINGO—a subset of the input language of the grounder GRINGO. The goal of this paper is to extend the syntax of terms permitted in that subset. First, instead of the specific choice of arithmetic functions made in earlier publications on mini-GRINGO, we approach integer arithmetic in an abstract way, so that different choices are allowed for different dialects of the language. Second, symbolic constants can be used now as function symbols. This generalization preserves the main property of the more limited form of the language established in earlier work: mini-GRINGO rules can be faithfully represented by first-order sentences.

1 Introduction

In answer set programming (ASP), a program verification task can be sometimes accomplished by transforming rules into first-order sentences so that the task is reduced to reasoning in classical logic, and invoking a resolution theorem prover. The proof assistant ANTHEM,[1] which implements this idea, allows us to reason about programs written in mini-GRINGO—a subset of the input language of the ASP grounder GRINGO.

The semantics of mini-GRINGO is defined in terms of the operator τ, which transforms a mini-GRINGO program into a "propositional theory"—a set of propositional formulas. Stable models (answer sets) of a propositional theory are defined by Paolo Ferraris [8]. Stable models of a mini-GRINGO program Π are defined as stable models of $\tau(\Pi)$ [14, Section 3].

If a program Π contains variables, then the set $\tau(\Pi)$ is infinite and thus cannot be directly used in computational procedures. ANTHEM relies instead on the operator τ^* [14, Section 6], which transforms a program into a finite set of first-order sentences. The target language of τ^* is two-sorted. There are *general* variables, which are similar to variables in mini-GRINGO programs: the set of values of a general variable includes both symbolic constants and (symbols for) integers. The other sort is *integer*, a subsort of *general*. The need to distinguish between general terms and integer terms when we represent rules by formulas

[1] `https://potassco.org/anthem/` .

is related to the difference between the treatment of function symbols in logic programming and in first-order logic. In an ASP program, the rule

```
foo(london + paris).
```

is syntactically correct, even though adding symbolic constants is not defined. (The stable model of this one-rule program is empty.) In first-order logic, on the other hand, functions represented by function symbols are required to be total. In the target language of τ^*, the expression $t_1 + t_2$ is an integer term if both t_1 and t_2 are integer terms. But if at least one of them is a general term then $t_1 + t_2$ is syntactically incorrect.

The claim that the translation τ^* faithfully represents the meaning of mini-GRINGO programs can be made precise using infinitary propositional formulas. Two-sorted first-order sentences can be transformed into infinitary propositional formulas by the process of grounding—replacing quantifiers by infinite conjunctions and disjunctions. For any mini-GRINGO program Π, the result of grounding $\tau^*(\Pi)$ is satisfied by the same HT-interpretations as $\tau(\Pi)$ [14, Proposition 3]. In other words, the result of grounding $\tau^*(\Pi)$ is strongly equivalent [11, 15] to $\tau(\Pi)$. It follows that both sets of formulas have the same stable models.

Some recent publications [5, 10, 13] extend mini-GRINGO by additional constructs found in many ASP programs—conditional literals and aggregates—and show how to extend the definition of τ^* to rules containing these constructs. The goal of this paper is to extend mini-GRINGO and τ^* in a different way: we extend the class of terms permitted in the language.

According to the Potassco User Guide, the input language of GRINGO includes symbols for 12 arithmetic functions on integers: addition, subtraction, unary minus, multiplication, integer division, modulo, exponentiation, absolute value, and four bitwise operations [9, Section 3.1.7]. The authors of the original publication on mini-GRINGO [14] chose to include the first six of these functions in the language and to disregard the other six. Including all twelve would not be difficult, but the technical results of the paper, strictly speaking, would need to be proved for the extended language anew. We address this difficulty by showing that integer arithmetic of mini-GRINGO can be described in an abstract way that covers many "dialects" of the language. In the definition proposed below, the set of arithmetic functions is a parameter. This is useful, in particular, because integer arithmetic is sometimes treated in different ways in different versions of Potassco software. In Version 6, the result of integer division will be understood as the quotient rounded toward negative infinity, whereas earlier versions round toward zero.[2]

In addition, the new version of mini-GRINGO allows symbolic constants to be used as uninterpreted function symbols [9, Section 3.1.1]. Such use of symbolic constants is common in answer set programming. For example, the blocks world planning program in the Potassco User Guide [9, Section 5.3] expresses that block `b1` is initially on the table by the rule

```
init(on(b1,table)).
```

[2] Roland Kaminski, personal communication (May 5, 2025).

Here `on` is an uninterpreted function symbol.

The main result of this paper shows that mini-GRINGO rules generalized as described above can be faithfully represented by first-order sentences: we define τ^* for the new version of the language in such a way that the result of grounding $\tau^*(\Pi)$ is strongly equivalent to $\tau(\Pi)$.

The syntax and semantics of extended mini-GRINGO are described in Section 2. The translation τ^* for the new version of the language is defined in Section 3. The main result is stated in Section 4, and a proof outline is presented in Section 5. Section 6 describes the direction of future work.

2 Mini-GRINGO

2.1 Syntax of terms

We assume that

– two disjoint sets of symbols are selected: *basic symbols* and *variables*; the latter is countably infinite;
– two disjoint sets of basic symbols are selected: *symbolic constants* and *numerals*; a 1-1 correspondence $n \mapsto \overline{n}$ between the set $\mathbf{Z}$ of integers and the set of numerals is selected;
– a set of *arithmetic function symbols* is selected, disjoint from the set of basic symbols and the set of variables; for every arithmetic function symbol f, a function $\hat{f}$ is selected that maps a subset of $\mathbf{Z}^k$ to $\mathbf{Z}$, where the *arity* k of f is a positive integer.

EXAMPLE. Symbolic constants are strings of letters, digits and underscores that begin with a lowercase letter. Variables are strings of letters, digits and underscores that begin with an upppercase letter. Numerals are (i) `0`*, (ii) strings of digits that begin with a digit different from* `0`*, and (iii)* `-` *followed by a string of type (ii). Basic symbols are symbolic constants, numerals,* `inf` *and* `sup`*. Arithmetic function symbols are*

$$+ \quad \times \quad /$$

and the corresponding functions $\hat{f}$ are addition, multiplication and integer division (floor of the quotient).

Mini-GRINGO *terms* are defined recursively:

– all basic symbols and variables are terms;
– if c is a symbolic constant and $\mathbf{t}$ is a non-empty tuple of terms, then $c(\mathbf{t})$ is a term;
– if f is an arithmetic function symbol of arity k, and $\mathbf{t}$ is a k-tuple of terms, then $f(\mathbf{t})$ is a term;
– if t_1 and t_2 are terms then $t_1 \mathinner{..} t_2$ is a term.

*EXAMPLE, CONTINUED. Abbreviations for mini-*GRINGO *terms:*

$$\begin{array}{rcl} -t & \text{is} & (\,-1)\times t \\ t_1 - t_2 & \text{is} & t_1 + (-\,t_2) \\ t_1 \setminus t_2 & \text{is} & t_1 - t_2 \times (t_1 \,/\, t_2) \end{array}$$

A mini-GRINGO term (or any other syntactic expression) is *ground* if it does not contain variables. A ground mini-GRINGO term is *precomputed* if it contains neither arithmetic function symbols nor the interval symbol. We assume that a total order on precomputed mini-GRINGO terms is selected such that numerals are contiguous (every term between two numerals is a numeral), and, for all integers m and n, $\overline{m} \leq \overline{n}$ iff $m \leq n$.

EXAMPLE, CONTINUED. `inf` *is smaller than numerals, which in turn are smaller than symbolic constants, which are smaller than complex terms, which are smaller than* `sup`. *Symbolic constants are ordered lexicographically. Complex terms are ordered both structurally and lexicographically, as described in the last clause of the definition of term ordering in the ASP-Core-2 document [1, Section 3].*

2.2 Syntax of programs

An *atom* is an expression of the form $c(\mathbf{t})$, where c is a symbolic constant and $\mathbf{t}$ is a tuple of mini-GRINGO terms (possibly empty, in which case the parentheses can be dropped). An atom $c(\mathbf{t})$ is *precomputed* if all members of $\mathbf{t}$ are precomputed mini-GRINGO terms. A *literal* is an atom possibly preceded by one or two occurrences of *not*.

Comparisons are expressions of the forms $t_1 \leq t_2$ and $t_1 \neq t_2$, where t_1 and t_2 are mini-GRINGO terms. Abbreviations:

$$\begin{array}{l} t_1 \geq t_2 \text{ stands for } t_2 \leq t_1, \\ t_1 = t_2 \text{ stands for } t_1 \leq t_2 \wedge t_1 \geq t_2, \\ t_1 < t_2 \text{ stands for } t_1 \leq t_2 \wedge t_1 \neq t_2, \\ t_1 > t_2 \text{ stands for } t_1 \geq t_2 \wedge t_1 \neq t_2. \end{array}$$

A *rule* is an expression of the form

$$H \leftarrow B_1 \wedge \cdots \wedge B_n \tag{1}$$

($n \geq 0$), where

- the head H is either an atom (then (1) is a *basic rule*), or an atom in braces (then (1) is a *choice rule*), or empty (then (1) is a *constraint*), and
- each member B_i of the body is a literal or a comparison.

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

2.3 Semantics of terms

The set $[t]$ of *values* of a ground mini-GRINGO term t is defined recursively:

- if t is a basic symbol then $[t]$ is $\{t\}$;
- if t is $c(t_1, \dots, t_k)$, where c is a symbolic constant, then $[t]$ is the set of terms of the form $c(s_1, \dots, s_k)$, where $s_i \in [t_i]$ $(i = 1, \dots, k)$;
- if t is $f(t_1, \dots, t_k)$, where f is an arithmetic function symbol, then $[t]$ is the set of numerals of the form $\overline{\hat{f}(n_1, \dots, n_k)}$, where $n_1, \dots, n_k$ is a tuple in the domain of $\hat{f}$ such that $\overline{n_i} \in [t_i]$ $(i = 1, \dots, k)$;
- if t is $t_1 \mathrel{..} t_2$ then $[t]$ is the set of numerals $\overline{n}$ for all integers n such that, for some integers n_1, n_2,

$$\overline{n_1} \in [t_1],\ \overline{n_2} \in [t_2],\ n_1 \le n \le n_2.$$

It is clear that values of a ground mini-GRINGO term are precomputed mini-GRINGO terms, and that $[t]$ is the singleton $\{t\}$ whenever t is precomputed.

EXAMPLE, CONTINUED.

$$[\mathsf{2}+\mathsf{2}] = \{\mathsf{4}\}, \quad [(\mathsf{1} \mathrel{..} \mathsf{2}) \times \mathsf{2}] = \{\mathsf{2}, \mathsf{4}\}, \quad [\mathsf{2} \,/\, \mathsf{0}] = [\mathtt{london} + \mathtt{paris}] = \emptyset.$$

For any ground mini-GRINGO terms $t_1 \dots, t_k$, by $[t_1, \dots, t_k]$ we denote the set of tuples $s_1, \dots, s_k$ of terms such that $s_1 \in [t_1], \dots,\ s_k \in [t_k]$.

2.4 Semantics of programs

The translation τ, defined below, transforms literals, comparisons and rules into propositional combinations of precomputed atoms. For any ground atom $c(\mathbf{t})$,

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

For any ground comparison $t_1 \prec t_2$, $\tau(t_1 \prec t_2)$ is

$\top$ (“true”), if there exist s_1 in $[t_1]$ and s_2 in $[t_2]$ such that $s_1 \prec s_2$;
$\bot$ (“false”), otherwise.

EXAMPLE, CONTINUED.

$$\begin{aligned}
&\tau(\mathtt{foo}((\mathsf{1} \mathrel{..} \mathsf{2}) \times \mathsf{2})) = \mathtt{foo}(\mathsf{2}) \vee \mathtt{foo}(\mathsf{4}),\\
&\tau(\mathtt{foo}(\mathtt{london} + \mathtt{paris})) = \bot,\\
&\tau(\mathtt{london} \le \mathtt{paris}) = \top.
\end{aligned}$$

If *Body* is a conjunction $B_1 \wedge B_2 \wedge \cdots$ of ground literals and ground comparisons then $\tau(\mathit{Body})$ stands for the conjunction $\tau(B_1) \wedge \tau(B_2) \wedge \cdots$.

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

$$\tau(\mathit{Body}) \to \bigwedge_{\mathbf{s}\in[\mathbf{t}]} c(\mathbf{s}).$$

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

$$\tau(\mathit{Body}) \to \bigwedge_{\mathbf{s}\in[\mathbf{t}]} (c(\mathbf{s}) \vee \neg c(\mathbf{s})).$$

If R is a ground constraint $\leftarrow \mathit{Body}$ then $\tau(R)$ is $\neg\tau(\mathit{Body})$.

An *instance* of a rule is a ground rule obtained from it by substituting precomputed mini-GRINGO terms for variables. For any program Π, $\tau(\Pi)$ is the set of the propositional formulas $\tau(R)$ for all instances R of the rules of Π.

A set of precomputed atoms is a *stable model* of a program Π if it is a stable model of $\tau(\Pi)$.

3 Representing terms and programs by formulas

3.1 Signature σ_0

An arithmetic function symbol f of arity k is *total* if the domain of $\hat{f}$ is the entire set $\mathbf{Z}^k$, and *partial* otherwise.

By σ_0 we denote the two-sorted signature that consists of

(i) the sort *general* and its subsort *integer*;
(ii) all numerals as object constants of sort *integer*;
(iii) all basic symbols other than numerals as object constants of sort *general*;
(iv) expressions $c\backslash k$, where c is a symbolic constant and k is a positive integer, as function constants of arity k, with both arguments and value of sort *general*;
(v) all total arithmetic function symbols as function constants with both arguments and value of sort *integer*;
(vi) expressions c/k, where c is a symbolic constant and k is a nonnegative integer, as predicate constants of arity k, with arguments of sort *general*;
(vii) the symbol $\leq$ as a binary predicate constant with both arguments of sort *general*.

(Partial arithmetic function symbols are not included, because they do not represent total functions even in application to integers.)

EXAMPLE, CONTINUED. Object constants of sort general *are symbolic constants,* `inf` *and* `sup`*. Function constants of group (v) are* $+$ *and* $\times$ *(but not* $/$*).*

We identify general variables with variables of mini-GRINGO.

3.2 Signature σ_0^- and the standard interpretation

By σ_0^- we denote the signature obtained from σ_0 by removing the predicate symbols c/k. The *standard interpretation* S of σ_0^- is defined as follows:

- its domain of the sort *general* is the set of all precomputed mini-GRINGO terms;
- its domain of the sort *integer* is the set of all numerals;
- if r is a basic symbol then $r^S = r$;
- for every function constant $c\backslash k$ and precomputed mini-GRINGO terms $t_1, \dots, t_k$,
$$(c\backslash k)^S(t_1, \dots, t_k) = c(t_1, \dots, t_k);$$
- for every total arithmetic function symbol f and integers $n_1, \dots, n_k$, where k is the arity of f,
$$f^S(\overline{n_1}, \dots, \overline{n_k}) = \overline{\hat{f}(n_1, \dots, n_k)};$$
- $\leq^S$ is the order on precomputed mini-GRINGO terms specified in the definition of mini-GRINGO (Section 2.1).

A ground term over σ_0 is *precomputed* if it does not contain arithmetic function symbols. It is clear that the map $r \mapsto r^S$ is a 1-1 correspondence between precomputed terms over σ_0 and precomputed mini-GRINGO terms; r^S can be obtained from r by replacing each function symbol $c\backslash k$ with c.

The set of sentences over σ_0^- that are satisfied by the standard interpretation is denoted by *Std*.

EXAMPLE, CONTINUED. The formula

$$\exists X \neg \exists N(N = X), \tag{2}$$

where X is a general variable and N is an integer variable, belongs to Std: take X to be `inf`*. By Lagrange's four-square theorem, the formula*

$$\forall N(N \geq \mathtt{0} \rightarrow \exists IJKL(N = I \times I + J \times J + K \times K + L \times L)),$$

where N, I, J, K, L are integer variables, belongs to Std as well.

Formula (2) is a sentence over the signature σ_0^- in every possible dialect of mini-GRINGO. But it does not necessarily belong to *Std* in dialects other than our running example, because the general framework of Section 2.1 allows the set of basic symbols to contain nothing other than numerals.

3.3 Representing mini-GRINGO terms by formulas

Assume that for every partial arithmetic function symbol f we chose a formula

$$V_f(N_1, \dots, N_{k+1})$$

over σ_0^-, where k is the arity of f, and $N_1, \dots, N_{k+1}$ are integer variables, with all free variables explicitly shown, which describes the function $\hat{f}$ in the following sense: for any integers $n_1, \dots, n_{k+1}$,

$$V_f(\overline{n_1}, \dots, \overline{n_{k+1}}) \in \mathit{Std} \text{ iff } \hat{f}(n_1, \dots, n_k) = n_{k+1}. \tag{3}$$

EXAMPLE, CONTINUED. The formula

$$(0 \leq N_1 - N_2 \times N_3 < N_2) \vee (0 \geq N_1 - N_2 \times N_3 > N_2)$$

can be chosen as $V_/$. *Indeed, the formula*

$$(0 \leq \overline{n_1} - \overline{n_2} \times \overline{n_3} < \overline{n_2}) \vee (0 \geq \overline{n_1} - \overline{n_2} \times \overline{n_3} > \overline{n_2})$$

is satisfied by the standard interpretation iff

$$0 \leq n_1 - n_2 \times n_3 < n_2 \tag{4}$$

or

$$0 \geq n_1 - n_2 \times n_3 > n_2. \tag{5}$$

Condition (4) is equivalent to the condition

$$n_2 > 0 \textit{ and } 0 \leq n_1/n_2 - n_3 < 1$$

and consequently to

$$n_2 > 0 \textit{ and } n_3 \leq n_1/n_2 < n_3 + 1.$$

Similarly, condition (5) is equivalent to the condition

$$n_2 < 0 \textit{ and } 0 \leq n_1/n_2 - n_3 < 1$$

and consequently to

$$n_2 < 0 \textit{ and } n_3 \leq n_1/n_2 < n_3 + 1.$$

For a mini-GRINGO term t and a general variable X that does not occur in t, $\mathit{val}_t(X)$ is the formula over σ_0 defined recursively:

- if t is a basic symbol or a variable then $\mathit{val}_t(X)$ is $X = t$;
- if t is $c(t_1, \dots, t_k)$, where c is a symbolic constant, then $\mathit{val}_t(X)$ is

  $$\exists X_1 \dots X_k(\mathit{val}_{t_1}(X_1) \wedge \dots \wedge \mathit{val}_{t_k}(X_k) \wedge X = (c\backslash k)(X_1, \dots, X_k)),$$

  where $X_1, \dots, X_k$ are general variables that do not occur in t;
- if t is $f(t_1, \dots, t_k)$, where f is a total arithmetic function symbol, then $\mathit{val}_t(X)$ is

  $$\exists N_1 \dots N_k(\mathit{val}_{t_1}(N_1) \wedge \dots \wedge \mathit{val}_{t_k}(N_k) \wedge X = f(N_1, \dots, N_k)),$$

  where $N_1, \dots, N_k$ are integer variables that do not occur in t;

- if t is $f(t_1, \dots, t_k)$, where f is a partial arithmetic function symbol, then $val_t(X)$ is

$$\exists N_1 \dots N_k N_{k+1}(val_{t_1}(N_1) \wedge \dots \wedge val_{t_k}(N_k) \wedge N_{k+1} = X \wedge \\ V_f(N_1, \dots, N_{k+1})),$$

  where $N_1, \dots, N_{k+1}$ are integer variables that do not occur in t;
- if t is $t_1 \,..\, t_2$ then $val_t(X)$ is

$$\exists N_1 N_2(val_{t_1}(N_1) \wedge val_{t_2}(N_2) \wedge N_1 \leq X \leq N_2),$$

  where N_1, N_2 are integer variables that do not occur in t.

The formula $val_t(X)$ expresses that X is a value of t, as defined in Section 2.3 for ground t; this is made precise in Lemma 3 (Section 5). It is clear that the free variables of this formula are X and the variables occurring in t.

EXAMPLE, CONTINUED. The formula $val_{c(Y)}(X)$ *is*

$$\exists X_1(X_1 = Y \wedge X = (c/1)(X_1)).$$

The formula $val_{Y+Z}(X)$ *is*

$$\exists N_1 N_2(N_1 = Y \wedge N_2 = Z \wedge X = N_1 + N_2).$$

The formula $val_{Y/Z}(X)$ *is*

$$\exists N_1 N_2 N_3(N_1 = Y \wedge N_2 = Z \wedge N_3 = X \\ \wedge ((0 \leq N_1 - N_2 \times N_3 < N_2) \vee (0 \geq N_1 - N_2 \times N_3 > N_2))).$$

3.4 Representing programs by formulas

The translation τ^B, defined below, produces a formula that characterizes the meaning of an expression in the body of a rule; this is made precise in Lemma 4 (Section 5). This translation transforms

- $c(t_1, \dots, t_k)$ into

$$\exists X_1 \cdots X_k(val_{t_1}(X_1) \wedge \dots \wedge val_{t_k}(X_k) \wedge (c/k)(X_1, \dots, X_k));$$

- *not* $c(t_1, \dots, t_k)$ into

$$\exists X_1 \cdots X_k(val_{t_1}(X_1) \wedge \dots \wedge val_{t_k}(X_k) \wedge \neg(c/k)(X_1, \dots, X_k));$$

- *not not* $c(t_1, \dots, t_k)$ into

$$\exists X_1 \cdots X_k(val_{t_1}(X_1) \wedge \dots \wedge val_{t_k}(X_k) \wedge \neg\neg(c/k)(X_1, \dots, X_k));$$

- $t_1 \prec t_2$ into $\exists X_1 X_2(val_{t_1}(X_1) \wedge val_{t_2}(X_2) \wedge X_1 \prec X_2)$;

where each X_i is a general variable.

If *Body* is a conjunction $B_1 \wedge B_2 \wedge \cdots$ of literals and comparisons then $\tau^B(\mathit{Body})$ stands for the conjunction $\tau^B(B_1) \wedge \tau^B(B_2) \wedge \cdots$.

The translation τ^* converts a basic rule

$$c(t_1, \dots, t_k) \leftarrow \mathit{Body}$$

of a program Π into the formula

$$\widetilde{\forall}(\mathit{val}_{t_1}(X_1) \wedge \cdots \wedge \mathit{val}_{t_k}(X_k) \wedge \tau^B(\mathit{Body}) \to (c/k)(X_1, \dots, X_k)),$$

where $X_1, \dots, X_k$ are alphabetically first general variables that do not occur in Π, and $\widetilde{\forall}$ denotes universal closure. A choice rule

$$\{c(t_1, \dots, t_k)\} \leftarrow \mathit{Body}$$

is converted into

$$\begin{aligned}\widetilde{\forall}(\mathit{val}_{t_1}(X_1) \wedge \cdots \wedge \mathit{val}_{t_k}(X_k) \wedge \tau^B(\mathit{Body}) \wedge \neg\neg(c/k)(X_1, \dots, X_k)&\\ \to (c/k)(X_1, \dots, X_k)),&\end{aligned}$$

and a constraint $\leftarrow \mathit{Body}$ becomes $\widetilde{\forall}\neg\tau^B(\mathit{Body})$.

By $\tau^*(\Pi)$ we denote the set of sentences $\tau^*(R)$ for all rules R of Π.

4 Relationship between Π and $\tau^*(\Pi)$

The relationship between Π and $\tau^*(\Pi)$ can be described using the grounding translation *gr*, which transforms sentences over σ_0 into infinitary propositional combinations of precomputed atoms. This translation is defined recursively:

- if F is $(c/k)(r_1, \dots, r_k)$ then $\mathit{gr}(F)$ is $c(r_1^S, \dots, r_k^S)$;
- if F is $r_1 \prec r_2$, where $\prec$ is $=$ or $\leq$, then $\mathit{gr}(F)$ is $\top$ if $r_1^S \prec r_2^S$, and $\bot$ otherwise;
- $\mathit{gr}(\bot)$ is $\bot$;
- $\mathit{gr}(F \odot G)$ is $\mathit{gr}(F) \odot \mathit{gr}(G)$ for every binary connective $\odot$;
- $\mathit{gr}(\forall X\, F(X))$ is the conjunction of the formulas $\mathit{gr}(F(r))$ over all precomputed terms r over σ_0 if X is a general variable, and over all numerals r if X is an integer variable;
- $\mathit{gr}(\exists X\, F(X))$ is the disjunction of the formulas $\mathit{gr}(F(r))$ over all precomputed terms r over σ_0 if X is a general variable, and over all numerals r if X is an integer variable.

(There is no clause for negation here because we treat $\neg F$ as shorthand for $F \to \bot$.)

Miroslaw Truszczynski [16] extended the stable model semantics of propositional theories proposed by Paolo Ferraris [8] to the infinitary case. Sets $\mathcal{H}_1$, $\mathcal{H}_2$ of infinitary formulas are *strongly equivalent* if, for every set $\mathcal{H}$ of infinitary formulas, the sets $\mathcal{H}_1 \cup \mathcal{H}$ and $\mathcal{H}_2 \cup \mathcal{H}$ have the same stable models [11, Section 3.1].

Strong equivalence of infinitary formulas can be characterized as equivalence in the deductive system denoted by HT^∞ [11, Corollary 2].[3]

Theorem. *For any mini-*GRINGO *program Π, $gr(\tau^*(\Pi))$ is strongly equivalent to $\tau(\Pi)$.*

Corollary. *For any mini-*GRINGO *program Π, $gr(\tau^*(\Pi))$ has the same stable models as Π.*

5 Proof outline

Lemma 1. *For any formula $F(X)$ over σ_0^- that has no free variables other than X,*

(a) if X is a general variable then
- *the sentence $\forall X\, F(X)$ belongs to Std iff for every precomputed term r over σ_0 the sentence $F(r)$ belongs to Std;*
- *the sentence $\exists X\, F(X)$ belongs to Std iff for at least one precomputed term r over σ_0 the sentence $F(r)$ belongs to Std;*

(b) if X is an integer variable then
- *the sentence $\forall X\, F(X)$ belongs to Std iff for every integer n the sentence $F(\overline{n})$ belongs to Std;*
- *the sentence $\exists X\, F(X)$ belongs to Std iff for a least one integer n the sentence $F(\overline{n})$ belongs to Std.*

Proof. Recall that a sentence belongs to *Std* iff it is satisfied by the standard interpretation S. (a) The domain of the sort *general* in S is the set of precomputed terms of mini-GRINGO, and every element of this domain is r^S for some precomputed term r over σ_0. (b) The domain of the sort *integer* in S is the set of numerals, and every element of this domain is $\overline{n}$ for some integer n.

Lemma 2. *For any sentence F over σ_0^-, $gr(F)$ is strongly equivalent to $\top$ if $F \in Std$, and to $\bot$ otherwise.*

The proof is by induction on the size of F, using Lemma 1.

Lemma 3. *Let $t(\mathbf{Z})$ be a mini-*GRINGO *term, where $\mathbf{Z}$ is a list of variables that contains every variable occurring in $t(\mathbf{Z})$, and let $F(\mathbf{Z}, X)$ stand for the formula $val_{t(\mathbf{Z})}(X)$. For any list $\mathbf{q}$ of precomputed terms over σ_0 of the same length as $\mathbf{Z}$ and any precomputed term r over σ_0, the formula $gr(F(\mathbf{q}, r))$ is strongly equivalent to $\top$ if $r^S \in [t(\mathbf{q}^S)]$, and to $\bot$ otherwise.*

Proof by induction on t. There are several cases to consider, depending on whether $t(\mathbf{Z})$ is a basic symbol (Case 1); a variable (Case 2); $c(t_1(\mathbf{Z}), \dots, t_k(\mathbf{Z}))$,

[3] This system is an infinitary version of the logic of here-and-there—the three-valued logic, intermediate between intuitionistic and classical, introduced by Arend Heyting [12], which plays an important role in the theory of stable models.

where c is a symbolic constant (Case 3); $f(t_1(\mathbf{Z}),\dots,t_k(\mathbf{Z}))$, where f is an arithmetic function symbol (Case 4); $t_1(\mathbf{q})\,..\,t_2(\mathbf{q})$ (Case 5). We consider here Case 4 as the most interesting. In this case, $t(\mathbf{q}^S)$ is $f(t_1(\mathbf{q}^S),\dots,t_k(\mathbf{q}^S))$; $[t(\mathbf{q}^S)]$ is the set of numerals $\overline{\hat{f}(n_1,\dots,n_k)}$, where $n_1,\dots,n_k$ is a tuple in the domain of $\hat{f}$ such that, for all i, $\overline{n_i} \in [t_i(\mathbf{q}^S)]$. The induction hypothesis is that for any precomputed terms $r_1,\dots,r_k$ over σ_0, $gr(F_i(\mathbf{q},r_i))$ is strongly equivalent to $\top$ if $r_i^S \in [t_i(\mathbf{q})]$, and to $\bot$ otherwise ($i=1,\dots,k$).

Case 4.1: f is total. The formula $F(\mathbf{Z},X)$ is

$$\exists N_1\dots N_k(F_1(\mathbf{Z},N_1)\wedge\dots\wedge F_k(\mathbf{Z},N_k)\wedge X=f(N_1,\dots,N_k)),$$

where $F_i(\mathbf{Z},X_i)$ is $val_{t(\mathbf{Z})}(X_i)$, so that $F(\mathbf{q},r)$ is

$$\exists N_1\dots N_k(F_1(\mathbf{q},N_1)\wedge\dots\wedge F_k(\mathbf{q},N_k)\wedge r=f(N_1,\dots,N_k)).$$

The formula $gr(F(\mathbf{q},r))$ is the disjunction of the conjunctions

$$gr(F_1(\mathbf{q},\overline{n_1}))\wedge\dots\wedge gr(F_k(\mathbf{q},\overline{n_k}))\wedge G_{n_1\cdots n_k} \tag{6}$$

where $n_1,\dots,n_k$ is an arbitrary tuple of integers, and $G_{n_1\cdots n_k}$ stands for $\top$ if

$$r^S=\overline{\hat{f}(n_1,\dots,n_k)}, \tag{7}$$

and for $\bot$ otherwise. Since r is precomputed, condition (7) can be rewritten as $r=\overline{\hat{f}(n_1,\dots,n_k)}$. *Case 4.1.1:* r is a numeral $\overline{n}$ for some integer n. Then $gr(F(\mathbf{q},r))$ is strongly equivalent to the disjunction of the formulas

$$gr(F_1(\mathbf{q},\overline{n_1}))\wedge\dots\wedge gr(F_k(\mathbf{q},\overline{n_k})),$$

where $n_1,\dots,n_k$ is an arbitrary tuple of integers such that

$$\hat{f}(n_1,\dots,n_k)=n. \tag{8}$$

By the induction hypothesis, this conjunction is srongly equivalent to $\top$ if

$$\overline{n_1}\in[t_1(\mathbf{q}^S)],\ \dots,\ \overline{n_k}\in[t_k(\mathbf{q}^S)], \tag{9}$$

and to $\bot$ otherwise. By (8), condition (9) is equivalent to $\overline{n}\in[t(\mathbf{q})]$. *Case 4.1.2:* r is not a numeral. Then $r^S\notin[t(\mathbf{q})]$, and all disjunctive terms (6) of $gr(F(\mathbf{q},r))$ are strongly equivalent to $\bot$.

Case 4.2: f is partial. The formula $F(\mathbf{Z},X)$ is

$$\exists N_1\dots N_{k+1}(F_1(\mathbf{Z},N_1)\wedge\dots\wedge F_k(\mathbf{Z},N_k)\wedge N_{k+1}=X\wedge V_f(N_1,\dots,N_{k+1})),$$

where $F_i(\mathbf{Z},X_i)$ is $val_{t(\mathbf{Z})}(X_i)$, so that $F(\mathbf{q},r)$ is

$$\exists N_1\dots N_{k+1}(F_1(\mathbf{q},N_1)\wedge\dots\wedge F_k(\mathbf{q},N_k)\wedge N_{k+1}=r\wedge V_f(N_1,\dots,N_{k+1})).$$

The formula $gr(F(\mathbf{q},r))$ is the disjunction of the conjunctions

$$gr(F_1(\mathbf{q},\overline{n_1})) \wedge \cdots \wedge gr(F_k(\mathbf{q},\overline{n_k})) \wedge G_{n_{k+1}} \wedge gr(V_f(\overline{n}_1,\ldots,\overline{n}_{k+1})) \tag{10}$$

where $n_1,\ldots,n_{k+1}$ is an arbitrary tuple of integers, and G_n stands for $\top$ if r is $\overline{n}$ and for $\bot$ otherwise. *Case 4.2.1: r is a numeral n.* By Lemma 2, it follows that $gr(F(\mathbf{q},r))$ is strongly equivalent to the disjunction of the conjunctions

$$gr(F_1(\mathbf{q},\overline{n_1})) \wedge \cdots \wedge gr(F_k(\mathbf{q},\overline{n_k}))$$

where $n_1,\ldots,n_k$ is an arbitrary tuple of integers satisfying the condition

$$V_f(\overline{n}_1,\ldots,\overline{n}_k,n) \text{ belongs to } \mathit{Std}.$$

By (3), this condition is equivalent to (8), and we can reason as in Case 4.1.1. *Case 4.2.2: r is not a numeral.* Then $r^S \notin [t(\mathbf{q})]$; all disjunctive terms (10) of $gr(F(\mathbf{q},r))$ are strongly equivalent to $\bot$, because the conjunctive term $G_{n_{k+1}}$ in each of them is $\bot$.

Lemma 4. *Let $\mathit{Body}(\mathbf{Z})$ be a conjunction of literals and comparisons, where $\mathbf{Z}$ is the list of all its variables, and let $F(\mathbf{Z})$ stand for the formula $\tau^B(\mathit{Body}(\mathbf{Z}))$. For any list $\mathbf{q}$ of precomputed terms over σ_0 of the same length as $\mathbf{Z}$, the formula $gr(F(\mathbf{q}))$ is strongly equivalent to $\tau(\mathit{Body}(\mathbf{q}^S))$.*

The proof uses Lemma 3.

In the proof of the theorem, it is sufficient to consider the case when the program consists of a single rule R. The proof uses Lemmas 3 and 4 and distinguishes between three cases, depending on whether R is a basic rule, a choice rule, or a constraint.

6 Future work

In published research on the original version of mini-GRINGO, the possibility of relating τ^* to τ is the basis of results on using τ^* for program verification [2, Theorem 3], [3, Section 7], [4, Theorems 1 and 2], [6, Theorem 2], [7, Theorem 3]. The result of this paper will allow us to extend this work to generalized mini-GRINGO and to extend the class of programs that can be verified by ANTHEM.

Acknowledgements

Thanks to Jorge Fandinno, Zachary Hansen and Yuliya Lierler for comments on a draft of this paper.

References

1. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., Schaub, T.: ASP-Core-2 input language format. Theory and Practice of Logic Programming **20**, 294–309 (2020)
2. Fandinno, J., Hansen, Z., Lierler, Y.: Axiomatization of aggregates in answer set programming. In: Proceedings of the AAAI Conference on Artificial Intelligence (2022)
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.: On Heuer's procedure for verifying strong equivalence. In: Proceedings of European Conference on Logics in Artificial Intelligence (2023)
5. Fandinno, J., Lifschitz, V.: Deductive systems for logic programs with counting: Preliminary report. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR) (2024)
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. Fandinno, J., Lifschitz, V., Temple, N.: Locally tight programs. Theory and Practice of Logic Programming (2024)
8. Ferraris, P.: Answer sets for propositional theories. In: Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). pp. 119–131 (2005)
9. 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/`
10. Hansen, Z., Lierler, Y.: Sm-based semantics for answer set programs containing conditional literals and arithmetic. In: Erdem, E., Vidal, G. (eds.) Practical Aspects of Declarative Languages - 27th International Symposium, PADL 2025, Denver, CO, USA, January 20-21, 2025, Proceedings. Lecture Notes in Computer Science, vol. 15537, pp. 71–87. Springer (2025)
11. Harrison, A., Lifschitz, V., Pearce, D., Valverde, A.: Infinitary equilibrium logic and strongly equivalent logic programs. Artificial Intelligence **246**, 22–33 (May 2017)
12. Heyting, A.: Die formalen Regeln der intuitionistischen Logik. Sitzungsberichte der Preussischen Akademie von Wissenschaften. Physikalisch-mathematische Klasse pp. 42–56 (1930)
13. Lifschitz, V.: Strong equivalence of logic programs with counting. Theory and Practice of Logic Programming **22** (2022)
14. 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)
15. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic **2**, 526–541 (2001)
16. 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)