Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

A Normal Form for Rules Containing Arithmetic Operations

Jorge Fandinno[1] , **Yuliya Lierler**[1] , **Vladimir Lifschitz**[2]
[1]University of Nebraska Omaha
[2]University of Texas at Austin
{jfandinno, ylierler}@unomaha.edu, vl@cs.utexas.edu

Abstract

This paper describes the process of translating rules that may contain arithmetic operations into the language of first-order logic. It identifies a normal form for which this transformation can be performed in a particularly simple and natural way. Other rules can be converted to this normal form by steps that preserve their meaning under the stable model semantics.

1 Introduction

This paper is about translating rules in the sense of answer set programming (ASP) (Marek and Truszczynski 1999; Niemelä 1999; Lifschitz 2019) into the language of first-order logic. Such translations help us clarify the relationship between logic programs and first-order theories as knowledge representation mechanisms. They are important also because of their role in the design of ANTHEM (Fandinno et al. 2025), a proof assistant for verifying ASP programs that operates by transforming verification tasks into first-order reasoning problems. We are interested in translations that are sound with respect to the stable model semantics, which has been defined both for ASP rules (Gebser et al. 2015) and for first-order formulas (Pearce and Valverde 2005; Ferraris, Lee, and Lifschitz 2011; Truszczynski 2012).

To illustrate some of the issues involved, compare the rule

$$q(X, Y+1) \leftarrow p(X, Y) \tag{1}$$

with its counterpart in the language of first-order logic:

$$\forall XY(p(X, Y) \rightarrow q(X, Y+1)).$$

When rule (1) is grounded, substituting symbolic constants for Y is not allowed, because Y occurs in the scope of an arithmetic operation (Calimeri et al. 2020, Section 3). To reflect this difference between X and Y in the formula above, we have to distinguish between variables of the sort *general*, such as X, and variables of the subsort *integer*, such as Y in this example. The syntax of the language requires that all arguments of an arithmetic operation be integer terms.

Consider now a rule containing the integer division operation:

$$q(X/Y+1) \leftarrow p(X, Y). \tag{2}$$

Since division is not a total function on domains that include zero, the expression X/Y with integer variables X, Y cannot be allowed in a first-order language: the semantics of these languages requires that every function symbol represent a total function. Transforming rule (2) into a formula has to be less straightforward and less natural than in the case of rule (1).

Translating rules containing placeholders (Gebser et al. 2019, Section 3.15) may be challenging in another way. Placeholders for integers are symbolic constants that are expected to be assigned numeric values before executing the program, and they may occur in the scope of an arithmetic operation. The fact $p(n+1)$, for instance is not a syntactically correct formula in the two-sorted language outlined above, because the first argument of addition in this expression is not an integer term. This fact can be represented by the longer formula

$$\forall I(I = n \rightarrow p(I+1)), \tag{3}$$

where I is an integer variable.

ANTHEM 2.0 implements two procedures for converting rules into formulas, τ^* and ν (Fandinno et al. 2025, Section 3.1). Each of them applies to rules in a fragment of the input language of the ASP grounder GRINGO (Gebser et al. 2019, Section 3.1), called mini-GRINGO, and generates formulas in a two-sorted first-order language. The translation τ^* (Lifschitz, Lühne, and Schaub 2019, Section 6) is applicable to all mini-GRINGO rules, whereas ν (Lifschitz 2021) is limited to rules satisfying some syntactic conditions. For example, ν is not applicable to rule (2). On the other hand, when ν is applicable, the result is often simpler and more natural than the result of applying τ^*. This makes ν more attractive for the user of ANTHEM, who often needs to read formulas representing rules.

In this paper, we propose a method of representing rules by formulas that combines the advantages of τ^* and ν: on the one hand, it is applicable to all mini-GRINGO rules; on the other, it often produces relatively simple translations. The process involves converting the given mini-GRINGO rule to a certain normal form. For example, the normal form of rule (2) is

$$q(Z+1) \leftarrow p(X, Y) \wedge Z = X/Y. \tag{4}$$

The normal form of the fact $p(n+1)$ is

$$p(X+1) \leftarrow X = n. \tag{5}$$

The normal form of the fact $p(1\,..\,8)$ ("p holds for all numbers between 1 and 8") is

$$p(X) \leftarrow X = 1\,..\,8. \tag{6}$$

Rules in normal form can be transformed into two-sorted formulas using a generalization of the translation ν. For example, the result of applying generalized ν to rule (5) is formula (3).

The role of the normal form described in this paper is determined by two theorems. First, if N is the normal form of a mini-GRINGO rule R then $\tau^*(N)$ is intuitionistically equivalent to $\tau^*(R)$. The fact that conversion to normal form can be justified without using the law of the excluded middle is essential because intuitionistically acceptable transformations are known to preserve stable models of a first-order theory (Pearce and Valverde 2005). Full classical logic does not have this property. For instance, if a first-order theory includes the axiom $\forall X(\neg p(X) \to q(X))$ then replacing that axiom by the classically equivalent formula $\forall X(\neg q(X) \to p(X))$ may change its stable models.

Second, the result of applying the generalized ν to any rule N in normal form is intuitionistically equivalent to $\tau^*(N)$.

In Sections 2 and 3 we review the syntax of mini-GRINGO rules and the definition of the translation τ^*. Our version of mini-GRINGO is more expressive than the class of programs studied in earlier work on transforming rules into formulas, because we allow symbolic constants to be used as function symbols. It is also more general than earlier work in the sense that we deal here with arbitrary dialects of mini-GRINGO (Lifschitz 2025), instead of one specific dialect. The process of converting rules to normal form is described in Section 4, and the generalized natural translation in Section 5. In Section 6 we show how formulas produced by the natural translation can be made completable. This additional transformation is required for constructing the completion of a mini-GRINGO program (Fandinno, Lifschitz, and Temple 2024, Section 3.1)—a step that plays an important role in the operation of ANTHEM (Fandinno et al. 2025, Section 3.2.2). Proofs are given in Sections 7 and 8.

2 Review: The Syntax of Mini-GRINGO

The syntax of rules is determined by a *mini-*GRINGO *signature*, which consists of

- a set of *basic symbols*, with some of them designated as *symbolic constants*,
- a countably infinite set of *variables*, disjoint from the set of basic symbols, and
- a set of *arithmetic function symbols*, with a positive integer, called the *arity*, assigned to each of them.

For example, we can define a mini-GRINGO signature Σ_0 by saying that symbolic constants are strings of letters and digits that begin with a lowercase letter; basic symbols are symbolic constants and integers (both positive and negative) in decimal notation; variables are strings of letters and digits that begin with an uppercase letter; the binary arithmetic function symbols are

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

*(Mini-*GRINGO*) terms* over a mini-GRINGO signature Σ are defined recursively:

- all basic symbols and variables of Σ are terms;
- if c is a symbolic constant of Σ, and $\mathbf{t}$ is a non-empty tuple of terms (separated by commas), then $c(\mathbf{t})$ is a term;
- if f is a k-ary arithmetic function symbol of Σ, and $\mathbf{t}$ is a k-tuple of terms, then $f(\mathbf{t})$ is a term.

A term is *precomputed* if it contains neither variables nor arithmetic function symbols. In other words, precomputed terms are formed from basic symbols using symbolic function symbols.

Atoms over Σ are symbolic constants and expressions of the form $c(\mathbf{t})$, where $\mathbf{t}$ is a non-empty tuple of terms. The expression $c()$ is alternative notation for the atom c. A *literal* over Σ is an atom possibly preceded by one or two occurrences of *not*. *Comparisons* over Σ are expressions of the form $t_1 \prec t_2$, where t_1 and t_2 are terms, and $\prec$ is one of the binary relation symbols

$$= \quad \neq \quad < \quad \leq \quad > \quad \geq$$

For a list $\mathbf{t}$ of terms $t_1, \dots, t_k$ and a list $\mathbf{t}'$ of terms $t'_1, \dots, t'_k$, the expression $\mathbf{t} = \mathbf{t}'$ stands for $t_1 = t'_1 \wedge \cdots \wedge t_k = t'_k$.

A *rule* over Σ is an expression of the form

$$H \leftarrow B_1 \wedge \cdots \wedge B_n \tag{7}$$

($n \geq 0$), where the head H is either an atom or empty, and each member B_i of the body is a literal or a comparison. Rules with the empty body are *facts*, and rules with the empty head are *constraints*. Rules of the form

$$p(\mathbf{X}) \leftarrow \mathbf{X} = \mathbf{t} \wedge B_1 \wedge \cdots \wedge B_n \wedge \textit{not not } p(\mathbf{X}),$$

where p is a symbolic constant, $\mathbf{t}$ is a list of terms, and $\mathbf{X}$ is a list of distinct variables of the same length as $\mathbf{t}$ that do not appear in $\mathbf{t}$, $B_1, \dots, B_n$, are called *choice rules* and can be written as

$$\{\, p(\mathbf{t}) \,\} \leftarrow B_1 \wedge \cdots \wedge B_n.$$

A *program over* Σ is a finite set of rules.

A *dialect* over a mini-GRINGO signature consists of

- a set of basic symbols called *numerals*, disjoint from the set of symbolic constants,
- a 1–1 correspondence $n \mapsto \overline{n}$ between the set $\mathbb{Z}$ of integers and the set of numerals,
- a total order on precomputed terms 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 $m \leq n$,
- for every arithmetic function symbol f, a function $\hat{f}$ that maps $\mathbb{Z}^k$ to the powerset of $\mathbb{Z}$, where k is the arity of f.[1]

For example, we can define a dialect D_0 of mini-GRINGO over the signature Σ_0 by saying that numerals are integers written in decimal notation; that $\overline{n}$ is the representation of n in decimal notation; that basic symbols are ordered as in the

[1] This description of $\hat{f}$ is different than in the paper by Lifschitz (2025). That version would not allow us to include the interval symbol .. among arithmetic function symbols.

language ASP-Core-2 (Calimeri et al. 2020, Section 3); and that the functions $\widehat{..}$, $\widehat{+}$, $\widehat{-}$ and $\widehat{\times}$ are defined by the formulas

$$\widehat{..}(m,n) = \{k : m \leq k \leq n\};$$
$$\widehat{+}(m,n) = \{m+n\}; \qquad \widehat{-}(m,n) = \{m-n\};$$
$$\widehat{\times}(m,n) = \{m \times n\};$$
$$\text{if } n \neq 0 \text{ then } \widehat{/}(m,n) = \{\lfloor m/n \rfloor\}; \quad \widehat{/}(m,0) = \emptyset.$$

In this example, every basic symbol is either a symbolic constant or a numeral. The signature Σ_0 can be extended by adding `#sup` and `#inf` (Gebser et al. 2019, Section 3.1.1) to the set of basic symbols; those would be neither symbolic constants nor numerals.

3 Review: Translation τ^*

3.1 Target Language of τ^*

An arithmetic function symbol f of a dialect of mini-GRINGO is *definite* if all values of $\hat{f}$ are singletons, and *indefinite* otherwise. In the dialect D_0, the symbols .. and / are indefinite (the latter because $\widehat{/}(m,0)$ is empty). This distinction is important because representing definite function symbols in a first-order language is particularly easy.

For any dialect D of mini-GRINGO, by $\sigma(D)$ we denote the two-sorted signature[2] that consists of

- the sort *general* and its subsort *integer*;
- all numerals of D as object constants of sort *integer*;
- all basic symbols of D other than numerals as object constants of sort *general*;
- expressions $c\backslash k$, where c is a symbolic constant of D and k is a positive integer, as k-ary function constants, with both arguments and value of sort *general*;
- all definite arithmetic function symbols of D as function constants with both arguments and value of sort *integer*;
- expressions c/k, where c is a symbolic constant of D and k is a nonnegative integer, as k-ary predicate constants with arguments of sort *general*;
- the symbol $\leq$ as a binary predicate constant with both arguments of sort *general*.

For example, the signature $\sigma(D_0)$ includes the function symbols $+$, $-$ and $\times$, but neither the division symbol nor the interval symbol.

Order relations other than $\leq$ are defined as abbreviations.

We identify variables of the sort *general* with variables of the dialect D.

3.2 Definable Symbols

By $\sigma^-(D)$ we denote the signature obtained from $\sigma(D)$ by removing all the predicate constants c/k. The *standard interpretation* S of $\sigma^-(D)$ is defined as follows:

- its domain of the sort *general* is the set of all precomputed terms of D;
- its domain of the sort *integer* is the set of all numerals of D;
- if r is a basic symbol of D then $r^I = r$;
- for every function constant $c\backslash k$ and precomputed terms $t_1, \dots, t_k$ of D,
$$(c\backslash k)^I(t_1, \dots, t_k) = c(t_1, \dots, t_k);$$
- for every definite arithmetic function symbol f of D and integers $n_1, \dots, n_k$, where k is the arity of f, $f^I(\overline{n_1}, \dots, \overline{n_k}) = \overline{n}$ for the integer n such that $\hat{f}(n_1, \dots, n_k) = \{n\}$;
- $\leq^I$ is the order relation of the dialect D.

The set of sentences over $\sigma^-(D)$ that are satisfied by the standard interpretation S is denoted by $Std(D)$.

Let f be a k-ary arithmetic function symbol of D, and let $F(I_1, \dots, I_{k+1})$ be a formula over the signature $\sigma^-(D)$ that contains neither symbolic object constants nor free variables other than the integer variables $I_1, \dots, I_{k+1}$. We say that the formula $F(I_1, \dots, I_{k+1})$ *defines* f if, for all integers $n_1, \dots, n_{k+1}$,

$$n_{k+1} \in \hat{f}(n_1, \dots, n_k) \text{ iff } F(\overline{n_1}, \dots, \overline{n_{k+1}}) \in Std(D). \quad (8)$$

If such a formula exists then we say that the symbol f is *definable*.

Every definite function symbol f is definable: we can take $F(I_1, \dots, I_{k+1})$ to be $f(I_1, \dots, I_k) = I_{k+1}$. Indeed, the formula $f(\overline{n_1}, \dots, \overline{n_k}) = \overline{n_{k+1}}$ is satisfied by the standard interpretation iff $\hat{f}(n_1, \dots, n_k) = \{n_{k+1}\}$, which is equivalent to the left-hand side of (8).

In the dialect D_0, all arithmetic function symbols are definable. Indeed, $+$, $-$ and $\times$ are definable because they are definite. The formula $I_1 \leq I_3 \leq I_2$ defines the interval symbol, and the formula

$$(0 \leq I_1 - I_2 \times I_3 < I_2) \vee (0 \geq I_1 - I_2 \times I_3 > I_2)$$

defines integer division.

In the rest of the paper, D is a dialect of mini-GRINGO with at least one symbolic constant, such that all its arithmetic function symbols are definable.

3.3 Values of a Term

For a mini-GRINGO term t and a general variable V that does not occur in t, the formula $val_t(V)$ over $\sigma(D)$, defined below, expresses that V is a value of t. In this definition, we use the following notation: If $\mathbf{t}$ and $\mathbf{V}$ are lists of terms $t_1, \dots, t_k$ and variables $V_1, \dots, V_k$ of the same length, respectively, then $val_{\mathbf{t}}(\mathbf{V})$ stands for the conjunction $val_{t_1}(V_1) \wedge \dots \wedge val_{t_k}(V_k)$. The definition is recursive:

- if t is a basic symbol or a variable then $val_t(V)$ is $V = t$;
- if t is $c(\mathbf{t})$, where c is a symbolic constant, then $val_t(V)$ is $\exists \mathbf{Y}(val_{\mathbf{t}}(\mathbf{Y}) \wedge V = (c\backslash k)(\mathbf{Y}))$, where $\mathbf{Y}$ is a list of general variables $Y_1, \dots, Y_k$ that do not occur in t;
- if t is $f(\mathbf{t})$, where f is a definite arithmetic function symbol, then $val_t(V)$ is $\exists \mathbf{I}(val_{\mathbf{t}}(\mathbf{I}) \wedge V = f(\mathbf{I}))$, where $\mathbf{I}$ is a list $I_1, \dots, I_k$ of integer variables;

[2] Syntax and semantics of many-sorted languages are reviewed by Fandinno, Lifschitz, and Temple (2024) in Appendix A.

$$\forall X_1 X_2 XY(X_1 = X \wedge \exists I_1 I_2(I_1 = Y \wedge I_2 = 1 \wedge X_2 = I_1 + I_2) \wedge (p/2)(X_1, X_2) \rightarrow (q/2)(X_1, X_2))$$
$$\forall X_1(\exists I_1 I_2 I_3(I_1 = 1 \wedge I_2 = 8 \wedge I_3 = X_1 \wedge I_1 \leq I_3 \leq I_2) \rightarrow (p/1)(X_1))$$

Figure 1: Formulas obtained by applying τ^* to rule (1) and to the fact $p(1..8)$.

- if t is $f(\mathbf{t})$, where f is an indefinite arithmetic function symbol, then $val_t(V)$ is

$$\exists \mathbf{I} I_{k+1}(val_{\mathbf{t}}(\mathbf{I}) \wedge I_{k+1} = V \wedge F(\mathbf{I}, I_{k+1})),$$

  where $\mathbf{I}$ is a list of integer variables $I_1, \ldots, I_k$, and $F(I_1, \ldots, I_{k+1})$ is a formula that defines f.

For example, the formula $val_X(V)$ is $V = X$. In the dialect D_0, $val_{Y+1}(V)$ is

$$\exists I_1 I_2(I_1 = Y \wedge I_2 = 1 \wedge V = I_1 + I_2),$$

and $val_{1\,..\,8}(V)$ is

$$\exists I_1 I_2 I_3(I_1 = 1 \wedge I_2 = 8 \wedge I_3 = V \wedge I_1 \leq I_3 \leq I_2).$$

3.4 Definition of τ^*

The translation τ^B produces a formula that characterizes the meaning of an expression in the body of a mini-GRINGO rule. It transforms

- an atom $c(\mathbf{t})$ into $\exists \mathbf{X}(val_{\mathbf{t}}(\mathbf{X}) \wedge (c/k)(\mathbf{X}))$;
- *not* $c(\mathbf{t})$ into $\exists \mathbf{X}(val_{\mathbf{t}}(\mathbf{X}) \wedge \neg(c/k)(\mathbf{X}))$;
- *not not* $c(\mathbf{t})$ into $\exists \mathbf{X}(val_{\mathbf{t}}(\mathbf{X}) \wedge \neg\neg(c/k)(\mathbf{X}))$;
- $t_1 \prec t_2$ into $\exists X_1 X_2(val_{t_1}(X_1) \wedge val_{t_2}(X_2) \wedge X_1 \prec X_2)$;

where $\mathbf{t}$ a list of terms $t_1, \ldots, t_k$, and $\mathbf{X}$ is a list of general variables $X_1, \ldots, X_k$ that do not occur in $\mathbf{t}$. Similarly, X_1 and X_2 in the last item are general variables that do not occur in t_1 and t_2.

For example, the result of applying τ^B to $p(X, Y)$ is

$$\exists X_1 X_2(X_1 = X \wedge X_2 = Y \wedge (p/2)(X_1, X_2)).$$

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

Now we are ready to define the translation τ^*. It converts a rule $c(\mathbf{t}) \leftarrow \mathit{Body}$ into the formula

$$\widetilde{\forall}(val_{\mathbf{t}}(\mathbf{X}) \wedge \tau^B(\mathit{Body}) \rightarrow (c/k)(\mathbf{X})),$$

where $\mathbf{t}$ and $\mathbf{X}$ are as in the definition of τ^B, and $\widetilde{\forall}$ denotes universal closure. A constraint $\leftarrow \mathit{Body}$ is converted onto $\widetilde{\forall} \neg \tau^B(\mathit{Body})$.

Two examples of applying τ^* are shown in Figure 1.

4 Normal Form

An *equation* is a comparison of the form $t_1 = t_2$. An equation is *numeric* if all basic symbols occurring in it are numerals.

A rule is *in normal form* if

(a) every basic symbol that occurs in it in the scope of an arithmetic function symbol is a numeral, and

(b) every occurrence of a term of the form $f(\mathbf{t})$ with indefinite f is the right-hand side of a numeric equation.

For example, all rules that do not contain arithmetic function symbols are in normal form. In the dialect D_0, rules (1) and (4)–(6) are in normal form. Rule (2) and the fact $p(1\,..\,8)$ are not in normal form because they violate condition (b). The facts $p(n+1)$ and $p(n(3)+1)$ violate condition (a).

We will show that any rule can be converted to normal form by applying, one or more times, the following transformation:

replace an occurrence of some term t by a fresh variable V and add the equation $V = t$ to the body of the rule. (9)

The algorithm for converting a rule to normal form uses the following terminology. About an occurrence of a term t in a rule we say that it is *abnormal* if

(a) it is in the scope of an arithmetic function symbol, the leading symbol[3] of t is a basic symbol, and t is not a numeral, or

(b) it is not the right-hand side of a numeric equation, and t has the form $f(\mathbf{t})$ with indefinite f.

A rule is in numeric normal form iff there are no abnormal occurrences of terms in it. This follows from the fact that any occurrence of a basic symbol in a rule in the scope of an arithmetic function symbol is the leading symbol of a term. Furthermore, an abnormal occurrence of a term in a rule is *innermost* if it has no abnormal subterms. For instance, in

$$q(1\,..\,(X/2)) \leftarrow p(X) \tag{10}$$

both $X/2$ and $1\,..\,(X/2)$ are abnormal; $X/2$ is innermost.

The normal form of a rule is constructed as follows:

while the rule contains an abnormal occurrence of a term
  choose an innermost abnormal occurrence;
  apply to it transformation (9).

For instance, in case of the fact $p(n+1)$, executing the algorithm replaces the abnormal occurrence n by the fresh variable X and adds the equation $X = n$ to the body of the rule, so that we get its normal form (5). For rule (10), the algorithm first replaces the innermost abnormal occurrence $X/2$ by a fresh variable Y and adds the equation $Y = X/2$ to the body of the rule, so that we get the intermediate result $q(1\,..\,Y) \leftarrow p(X) \wedge Y = X/2$. It still contains the abnormal occurrence $1\,..\,Y$. The second iteration of the algorithm replaces that occurrence by a fresh variable Z and

[3] The *leading symbol* of a term is the term itself if it is a basic symbol or a variable; the symbolic constant c if it is $c(\mathbf{t})$; and the arithmetic function symbol f if it is $f(\mathbf{t})$.

adds the equation $Z = 1\,..\,Y$ to the body of the rule, so that we get its normal form

$$q(Z) \leftarrow p(X) \wedge Y = X/2 \wedge Z = 1\,..\,Y.$$

Rule

$$p(X) \leftarrow n = X/2 \tag{11}$$

has the abnormal occurrence $X/2$; executing the algorithm replaces it by a fresh variable Y and adds $Y = X/2$ to the body of the rule, so that we get its normal form

$$p(X) \leftarrow n = Y \wedge Y = X/2.$$

The algorithm above is guaranteed to terminate, because term replacements do not introduce new abnormal occurrences, and the occurrence that is replaced at any step will not be abnormal anymore. Indeed, consider the effect of such a replacement. The abnormal occurrence of t that was replaced becomes the right-hand side of an equation $V = t$. If the occurrence was of type (a), then it is not abnormal anymore, because in the new position it is not in the scope of an arithmetic function symbol. Otherwise it was of type (b), so that t has the form $f(\mathbf{t})$, where f is an arithmetic function symbol. Since this was an innermost abnormal occurrence, $f(\mathbf{t})$ did not contain abnormal subterms of type (a). It follows that all basic symbols occurring in $f(\mathbf{t})$ are numerals, so that the equation $V = t$ is numeric. So in this case t in the new position is not abnormal either.

The theorem below describes the relationship between a rule and its normal form. By $\mathit{Int}(D)$ we denote the many-sorted intuitionistic predicate calculus with equality (Fandinno and Lifschitz 2023, Section 5.1) over the signature $\sigma(D)$.

Theorem 1. *If N is the normal form of a rule R then $\tau^*(N)$ is equivalent to $\tau^*(R)$ in* Int(D).

5 Generalized Natural Translation

Consider a rule N in normal form. Applying the translation ν to N involves substituting an integer variable for each variable that occurs in N at least once

- in the scope of an arithmetic function symbol, or
- in the left-hand side of an equation that contains an indefinite function symbol in the right-hand side.

Make the list $V_1, \dots, V_m$ of all variables satisfying this condition, and choose m distinct integer variables $I_1, \dots, I_m$. For any mini-GRINGO term t that occurs in N and does not contain indefinite function symbols, the result of

(i) substituting $I_1, \dots, I_m$ for $V_1, \dots, V_m$ and

(ii) replacing every subexpression of the form $c(r_1, \dots, r_k)$, where c is a symbolic constant, with $(c\backslash k)(r_1, \dots, r_k)$

is a term over the signature $\sigma(D)$. The operator that performs this transformation will be denoted by $p2f$. It applies to tuples of terms componentwise.

For example, if N is rule (1) then $m = 1$, V_1 is Y, and $p2f(X, Y + 1)$ is $(X, I_1 + 1)$. If N is rule (4) then $m = 3$, $p2f(X, Y)$ is (I_1, I_2), and $p2f(Z + 1)$ is $I_3 + 1$.

The translation ν is defined first for expressions that may occur in the head and the body of N:

- If $\mathbf{t}$ is a k-tuple of terms that do not contain indefinite function symbols then
  - $\nu(p(\mathbf{t}))$ is $(p/k)(p2f(\mathbf{t}))$,
  - $\nu(\mathit{not}\ p(\mathbf{t}))$ is $\neg(p/k)(p2f(\mathbf{t}))$,
  - $\nu(\mathit{not\ not}\ p(\mathbf{t}))$ is $\neg\neg(p/k)(p2f(\mathbf{t}))$.
- If $t_1 \prec t_2$ is a comparison that does not contain indefinite function symbols then $\nu(t_1 \prec t_2)$ is $p2f(t_1) \prec p2f(t_2)$.
- $\nu(t = f(t_1, \dots, t_k))$, where f is an indefinite function symbol, is $F\big(p2f(t_1), \dots, p2f(t_k), p2f(t)\big)$, where $F(N_1, \dots, N_{k+1})$ is a formula defining f.
- The result of applying ν to the empty string is $\bot$ (false).

The result of applying the translation ν to a rule (7) in normal form is defined as the sentence

$$\widetilde{\forall}(\nu(B_1) \wedge \cdots \wedge \nu(B_n) \rightarrow \nu(H)). \tag{12}$$

For example, ν transforms rule (1) into

$$\forall X I_1((p/2)(X, I_1) \rightarrow (q/2)(X, I_1{+}1)), \tag{13}$$

rule (5) into

$$\forall I_1(I_1 = n \rightarrow (p/1)(I_1)),$$

and rule (6) into

$$\forall I_1(1 \leq I_1 \leq 8 \rightarrow (p/1)(I_1)).$$

The theorem below shows that the translations ν and τ^* are equivalent whenever the former is applicable. It generalizes a theorem due to Lifschitz (2021).

Theorem 2. *For any rule N in normal form, the formula $\nu(N)$ is equivalent to $\tau^*(N)$ in* Int(D).

6 Making the Translation Completable

A sentence over $\sigma(D)$ is *completable* if it has the form

$$\widetilde{\forall}(F \rightarrow (p/k)(\mathbf{V})), \tag{14}$$

where $\mathbf{V}$ is a list of pairwise distinct general variables. This is a special case of the definition used by Fandinno, Lifschitz, and Temple (2024) for extending program completion (Clark 1978) to mini-GRINGO programs. Applying the transformation τ^* to any non-constraint rule produces a completable sentence.

The sentence obtained by applying ν to a non-constraint rule is not always completable, but it can be made completable by intuitionistically equivalent transformations as follows. This sentence has the form

$$\widetilde{\forall}(F \rightarrow (p/k)(t_1, \dots, t_k)).$$

Pick new general variables $V_1, \dots, V_k$. For $i = 1, \dots, k$, if t_i is not a general variable different from $t_1, \dots, t_{i-1}$, then
replace t_i in the consequent by V_i;
add the conjunctive term $V_i = t_i$ to the antecedent;
add $\forall V_i$ to the list of quantifiers in front of the implication.

It is clear that after applying this transformation all terms in the consequent are pairwise distinct general variables, so

that the result is completable. It is clear also that each step is an intuitionistically equivalent transformation.

For example, the result (13) of applying ν to rule (1) is not completable, because the term $I_1 + 1$ is not a variable. Applying the transformation above to (13) gives the completable sentence

$$\forall XV_2I_1((p/2)(X, I_1)\wedge V_2 = I_1+1 \rightarrow (q/2)(X, V_2)). \quad (15)$$

To give an example of using the algorithm above for forming the completed definition of a predicate (Fandinno, Lifschitz, and Temple 2024, Section 3.1), consider a program that defines the predicate $q/2$ by two rules: (1) and

$$q(X, Y) \leftarrow r(Y, X).$$

The last rule is in normal form, and the result of applying ν to it is the completable sentence

$$\forall XY((r/2)(Y, X) \rightarrow (q/2)(X, Y)). \quad (16)$$

The completed definition of $q/2$ can be formed now in two steps. First, we rename bound variables in (15) and (16) so that the consequents of the implications become identical. For example, this can be accomplished by rewriting (15) as

$$\forall XYI_1((p/2)(X, I_1) \wedge Y = I_1 + 1 \rightarrow (q/2)(X, Y)).$$

Second, we form an equivalence expressing that, jointly, these two sufficient conditions for $q/2$ are necessary:

$$\forall XY((q/2)(X, Y) \leftrightarrow \exists I_1((p/2)(X, I_1) \wedge Y = I_1 + 1) \vee (r/2)(Y, X)).$$

7 Proof of Theorem 1

In the following, we say that two formulas F and G are *equivalent* if $F \leftrightarrow G$ is provable in $Int(D)$. We say that two rules are equivalent if the sentences obtained by applying τ^* to them are equivalent formulas. Theorem 1 follows from the fact that applying procedure (9) to any rule results in an equivalent rule (Lemma 4 below).

We introduce some notation. We use $E[V]$ to indicate that E is a formula, term, or other syntactic entity containing at most one occurrence of the variable V. By $E[t]$ we denote the result of replacing that occurrence of V in $E[V]$ by t.

Lemma 1. *If $r[V]$ is a term with exactly one occurrence of a variable V, and t is a term that does not contain V, then*

$$val_{r[t]}(W) \leftrightarrow \exists V\,(val_{r[V]}(W) \wedge val_t(V)) \quad (17)$$

is provable in $Int(D)$.

Proof. By induction on $r[V]$. *Basis:* $r[V]$ is V, and (17) is

$$val_t(W) \leftrightarrow \exists V\,(W = V \wedge val_t(V)),$$

which is equivalent to

$$val_t(W) \leftrightarrow \exists V\,(W = V \wedge val_t(W)).$$

Since V does not occur in $val_t(W)$, the last formula is provable in $Int(D)$. *Induction step:* $r[V]$ has the form $c(\mathbf{r})$, where c is a symbolic constant, or $f(\mathbf{r})$, where f is an arithmetic function symbol. Assume, for instance, that V occurs in the first member of the tuple $\mathbf{r}$. *Case 1:* $r[V]$ is $c(r_1[V], r_2, \ldots, r_k)$. Then the left-hand side and right-hand side of (17) respectively are

$$\exists \mathbf{Y}\,(val_{r_1[t]}(Y_1)\ \wedge G \wedge W = P), \quad (18)$$

$$\exists V(\exists \mathbf{Y}\,(val_{r_1[V]}(Y_1) \wedge G \wedge W = P) \wedge val_t(V)), \quad (19)$$

where $\mathbf{Y}$ is a list $Y_1, \ldots, Y_k$ of variables, G stands for

$$val_{r_2}(Y_2) \wedge \cdots \wedge val_{r_k}(Y_k),$$

and P stands for $(c\backslash k)(Y_1, \ldots, Y_k)$. Since V occurs neither in G nor in P, formula (19) is equivalent to

$$\exists \mathbf{Y}\,(\exists V\,(val_{r_1[V]}(Y_1) \wedge val_t(V)) \wedge G \wedge W = P). \quad (20)$$

By the induction hypothesis,

$$val_{r_1[t]}(W_1) \leftrightarrow \exists V\,(val_{r_1[V]}(W_1) \wedge val_t(V))$$

is provable in $Int(D)$, and thus (20) is equivalent to (18). *Case 2:* $r[V]$ is $f(r_1[V], \ldots, r_k)$. If f is definite then the left-hand side and right-hand side of (17) respectively are

$$\exists \mathbf{Y}\,(val_{r_1[t]}(Y_1)\ \wedge G \wedge W = f(\mathbf{Y}),$$
$$\exists V(\exists \mathbf{Y}\,(val_{r_1[V]}(Y_1) \wedge G \wedge W = f(\mathbf{Y})) \wedge val_t(V)),$$

where G is as above. The rest of the proof is analogous to Case 1. If f is indefinite then the left-hand side and right-hand side of (17) respectively are

$$\exists \mathbf{Y}(val_{r_1[t]}(Y_1)\ \wedge G \wedge I_{k+1} = W \wedge F(I_1, \ldots, I_{k+1})),$$
$$\exists V(\exists \mathbf{Y}(val_{r_1[V]}(Y_1) \wedge G \wedge I_{k+1} = W \wedge F(I_1, \ldots, I_{k+1})) \wedge val_t(V)).$$

The rest of the proof is analogous to Case 1. □

Lemma 2. *If t is a term and V is a variable, then*

$$\tau^B(V = t) \leftrightarrow val_t(V)$$

is provable in $Int(D)$.

Proof. Note that $\tau^B(V = t)$ is

$$\exists X_1X_2(X_1 = V \wedge val_t(X_2) \wedge X_1 = X_2)$$

where X_1 and X_2 are fresh variables. This is equivalent to

$$\exists X_1X_2(X_1 = V \wedge val_t(V) \wedge V = X_2),$$

which is equivalent to $val_t(V)$ because X_1, X_2 do not occur in $val_t(V)$. □

Lemma 3. *If $L[V]$ is a mini-*GRINGO *literal or comparison containing exactly one occurrence of V, and t is a term that does not contain V, then*

$$\tau^B(L[t]) \leftrightarrow \exists V(\tau^B(L[V]) \wedge \tau^B(V = t)) \quad (21)$$

is provable in $Int(D)$.

Proof. Assume, for instance, that $L[V]$ is an atom $c(r_1[V], r_2, \ldots, r_k)$. Then the left-hand side and right-hand side of (21) respectively are

$$\exists \mathbf{Y}\,(val_{r_1[t]}(Y_1)\ \wedge G \wedge P), \quad (22)$$

$$\exists V(\exists \mathbf{Y}\,(val_{r_1[V]}(Y_1) \wedge G \wedge P) \wedge \tau^B(V = t)), \quad (23)$$

where G and P are as in the proof of Lemma 1. By Lemma 2, formula (23) is equivalent to

$$\exists V(\exists \mathbf{Y}\,(val_{r_1[V]}(Y_1) \wedge G \wedge P) \wedge val_t(V)). \tag{24}$$

Furthermore, since none of $Y_1, \ldots, Y_k$ occur free in $val_t(V)$, and V occurs neither in G nor P, formula (24) is equivalent to

$$\exists \mathbf{Y}\,(\exists V(val_{r_1[V]}(Y_1) \wedge val_t(V)) \wedge G \wedge P)$$

By Lemma 1, the last formula is equivalent to (22). The other cases are similar. □

Lemma 4. *For any mini-*GRINGO *rule of the form*

$$Head[V] \leftarrow Body[V]$$

with exactly one occurrence of variable V, and any term t that does not contain V, the equivalence between the sentences

$$\tau^*(Head[t] \leftarrow Body[t]), \tag{25}$$

$$\tau^*(Head[V] \leftarrow Body[V] \wedge V = t) \tag{26}$$

is provable in $Int(D)$.

Proof. We proceed by cases based on the position of the single occurrence of V. Assume first that the single occurrence of V is in $Head[V]$. Then, $Body[V]$ and $Body[t]$ are identical, and $Head[V]$ is of the form $c(r_1, \ldots, r_k)$ with V occurring in $r_i[V]$ for some $i \in \{1, \ldots, k\}$. Assume, for instance, that $i = 1$. Sentences (25) and (26) can be respectively written as the universal closures of formulas

$$val_{r_1[t]}(Y_1) \wedge B \rightarrow H \tag{27}$$

$$val_{r_1[V]}(Y_1) \wedge B \wedge \tau^B(V = t) \rightarrow H \tag{28}$$

where H stands for $(c/k)(Y_1, \ldots, Y_k)$ and B stands for $\tau^B(Body[V]) \wedge G$ with G as in the proof of Lemma 1. Since the only occurrence of V in the rule is in $r_1[V]$, it follows that V does not occur in B nor H. Hence, (28) is equivalent to

$$\exists V(val_{r_1[V]}(Y_1) \wedge \tau^B(V = t)) \wedge B \rightarrow H \tag{29}$$

which, by Lemma 2, is equivalent to

$$\exists V(val_{r_1[V]}(Y_1) \wedge val_t(V)) \wedge B \rightarrow H \tag{30}$$

By Lemma 1, this is equivalent to (27).

Assume now that the single occurrence of V is in $Body[V]$. Then, $Head[V]$ and $Head[t]$ are identical and $Body[V]$ is of the form $L_1 \wedge \cdots \wedge L_n$ with V occurring in $L_i[V]$ for some $i \in \{1, \ldots, n\}$. Assume, for instance, that $i = 1$. Sentences (25) and (26) can be respectively written as the universal closures of formulas

$$\tau^B(L_1[t]) \wedge B \rightarrow H \tag{31}$$

$$\tau^B(L_1[V]) \wedge B \wedge \tau^B(V = t) \rightarrow H \tag{32}$$

where H is as above, and B stands for

$$\tau^B(L_2) \wedge \cdots \wedge \tau^B(L_n).$$

Since the only occurrence of V in the rule is in $L_i[V]$, it follows that V does not occur in B nor H. Hence, (32) is equivalent to

$$\exists V(\tau^B(L_1) \wedge \tau^B(V = t)) \wedge B \rightarrow H$$

which, by Lemma 3, is equivalent to (31). □

8 Proof of Theorem 2

Assume that N is a rule (7) in normal form, and that C is

$$I_1 = V_1 \wedge \cdots \wedge I_m = V_m,$$

where $V_1, \ldots, V_m, I_1, \ldots, I_m$ are variables as in the definition of $p2f$ (Section 5).

Lemma 5. *For any term t that occurs in N in the scope of an arithmetic function or in an equation of the form $t = t'$ such that t' contains an indefinite function symbol, $p2f(t)$ is of the sort integer.*

Proof. Case 1: t is a variable. Since N is in normal form and t occurs in the scope of a definite arithmetic function symbol or in an equation of the form $t = t'$ such that t' contains an indefinite function symbol, t is one of the variables V_k ($1 \leq k \leq m$). Then $p2f(V_k)$ is the integer variable I_k.

Case 2: t is a basic symbol. Then $p2f(t)$ is t. By the definition of normal form, t is a numeral.

Case 3: t is of the form $f(t_1, \ldots, t_k)$. From condition (b) in the definition of normal form, f is a definite arithmetic function symbol. Then $p2f(f(t_1, \ldots, t_k))$ is $f(p2f(t_1), \ldots, p2f(t_k))$, which is an integer term. □

Lemma 6. *For any tuple $\mathbf{t}$ of terms that occur in N and contain no occurrences of indefinite function symbols, and any tuple $\mathbf{V}$ of variables of the same length as $\mathbf{t}$, the formulas*

(i) $C \rightarrow \forall \mathbf{V}(val_{\mathbf{t}}(\mathbf{V}) \leftrightarrow \mathbf{V} = p2f(\mathbf{t}))$;

(ii) $C \rightarrow (\nu(p(\mathbf{t})) \leftrightarrow \forall \mathbf{V}(val_{\mathbf{t}}(\mathbf{V}) \rightarrow (p/k)(\mathbf{V})))$, *where k is arity of $\mathbf{t}$;*

(iii) $C \rightarrow (\nu(p(\mathbf{t})) \leftrightarrow \tau^B(p(\mathbf{t})))$;

(iv) $C \rightarrow (\nu(not\ p(\mathbf{t})) \leftrightarrow \tau^B(not\ p(\mathbf{t})))$;

(v) $C \rightarrow (\nu(not\ not\ p(\mathbf{t})) \leftrightarrow \tau^B(not\ not\ p(\mathbf{t})))$

are provable $Int(D)$.

Proof. (i) It is sufficient to consider the case when $\mathbf{t}$ is a single term t, so that the formula to be proven is

$$C \rightarrow \forall V(val_t(V) \leftrightarrow V = p2f(t)). \tag{33}$$

The proof is by induction on t. *Case 1:* t is one of the variables V_k ($1 \leq k \leq m$). Then the consequent of (33) is $\forall V(V = V_k \leftrightarrow V = I_k)$, and the antecedent C contains the conjunctive term $V_k = I_k$.

Case 2: t is a variable different from $V_1, \ldots, V_m$, or a basic symbol. Then the consequent of (33) is

$$\forall V(V = t \leftrightarrow V = t).$$

Case 3: t is $c(\mathbf{t})$, where c is a symbolic constant and $\mathbf{t}$ is a list $t_1, \ldots, t_k$ of terms. Then $val_t(V)$ is

$$\exists \mathbf{Y}(val_{\mathbf{t}}(\mathbf{Y}) \wedge V = (c\backslash k)(\mathbf{Y}))$$

where $\mathbf{Y}$ is a list $Y_1, \ldots, Y_k$ of general variables not occurring in t. By the induction hypothesis, under the assumption C, the formula above can be rewritten as

$$\exists \mathbf{Y}(\mathbf{Y} = p2f(\mathbf{t}) \wedge V = (c\backslash k)(\mathbf{Y})),$$

which is equivalent to

$$V = (c\backslash k)(p2f(\mathbf{t})).$$

The right-hand side is $p2f(c(\mathbf{t}))$.

Case 4: t is $f(\mathbf{t})$, where f is a definite arithmetic function symbol, and $\mathbf{t}$ is a list $t_1, \dots, t_k$ of terms. Then $val_t(V)$ is

$$\exists \mathbf{J}(val_{\mathbf{t}}(\mathbf{J}) \wedge V = f(\mathbf{J}))$$

where $\mathbf{J}$ is a list $J_1, \dots, J_k$ of integer variables. Then, by the induction hypothesis, under the assumption C, the formula above can be rewritten as

$$\exists \mathbf{J}(\mathbf{J} = p2f(\mathbf{t}) \wedge V = f(\mathbf{J})).$$

By Lemma 5, each of the terms in the list $p2f(\mathbf{t})$ is of the sort integer. Hence the above formula is equivalent to

$$V = f(p2f(\mathbf{t})).$$

Thus under the assumption C,

$$val_{f(\mathbf{t})}(V) \leftrightarrow V = f(p2f(\mathbf{t}))$$

It remains to observe that $f(p2f(\mathbf{t}))$ is $p2f(f(\mathbf{t}))$.

Parts (ii)–(v) follow as in the proof of Lemma 1 (ii)-(v) by Lifschitz (2021). □

Lemma 7. *For any comparison $t_1 \prec t_2$ that occurs in N and contains no occurrences of indefinite function symbols, the formula*

$$C \to (\nu(t_1 \prec t_2) \leftrightarrow \tau^B(t_1 \prec t_2))$$

is provable in $Int(D)$.

Proof. The proof uses Lemma 6(i) and is similar to the proof of Lemma 2 by Lifschitz (2021). □

By $\mathbf{J}$ we denote a list $J_1, \dots, J_k$ of integer variables.

Lemma 8. *Let f be an indefinite function symbol, and let $F(\mathbf{J}, K)$ be a formula over $\sigma^-(D)$ defining f. For any term $f(\mathbf{t})$ occurring in N and every term r of the sort integer, the formula*

$$C \to (val_{f(\mathbf{t})}(r) \leftrightarrow F(p2f(\mathbf{t}), r)) \tag{34}$$

is provable in $Int(D)$.

Note that substituting $p2f(\mathbf{t})$ for $\mathbf{J}$ in $F(\mathbf{J}, K)$ is allowed because, by Lemma 5, $p2f(\mathbf{t})$ is a tuple of integer terms.

Proof. The left-hand side of the equivalence in (34) stands for

$$\exists \mathbf{J} K(val_{\mathbf{t}}(\mathbf{J}) \wedge K = r \wedge F(\mathbf{J}, K))$$

which is equivalent to

$$\exists \mathbf{J}(val_{\mathbf{t}}(\mathbf{J}) \wedge F(\mathbf{J}, r)), \tag{35}$$

because the term r is of the sort integer. By condition (b) in the definition of normal form, the terms $\mathbf{t}$ do not contain indefinite function symbols. By Lemma 6(i), formula (35) can be rewritten, under the assumption C, as

$$\exists \mathbf{J}(\mathbf{J} = p2f(\mathbf{t}) \wedge F(\mathbf{J}, r)).$$

This is equivalent to the right-hand side of the equivalence in (34). □

Lemma 9. *For any numeric equation $t = f(\mathbf{t})$ occurring in N, where f is an indefinite function symbol, the formula*

$$C \to (\nu(t = f(\mathbf{t})) \leftrightarrow \tau^B(t = f(\mathbf{t})))$$

is provable in $Int(D)$.

Proof. Note that $\tau^B(t = f(\mathbf{t}))$ is

$$\exists X_1 X_2 \big(val_t(X_1) \wedge val_{f(\mathbf{t})}(X_2) \wedge X_1 = X_2\big)$$

where X_1 and X_2 are general variables. This formula is equivalent to

$$\exists X \big(val_t(X) \wedge val_{f(\mathbf{t})}(X)\big).$$

By Lemma 6(i), the last formula can be rewritten, under the assumption C, as

$$\exists X \big(X = p2f(t) \wedge val_{f(\mathbf{t})}(X)\big),$$

which can be further rewritten as

$$val_{f(\mathbf{t})}(p2f(t)).$$

By Lemma 5, $p2f(t)$ is an integer term. By Lemma 8, the last formula can be rewritten, under the assumption C, as

$$F(p2f(\mathbf{t}), p2f(t)),$$

which is identical to $\nu(t = f(\mathbf{t}))$. □

Lemma 10. *If B is a literal or a comparison occurring in the body of N, then the formula*

$$C \to (\nu(B) \leftrightarrow \tau^B(B))$$

is provable in $Int(D)$.

Proof. The assertion of the lemma is given by Lemma 6(iii)–(v) for literals, by Lemma 7 for comparisons without occurrences of indefinite function symbols, and by Lemma 9 for numeric equations cointaining an indefinite function symbol. □

Lemma 11. *If a term t occurs in the rule N, and V is a variable that occurs in t at least once in the scope of an arithmetic function symbol, then the formula*

$$\exists W\, val_t(W) \to \exists I(I = V),$$

where W is a general variable, is provable in $Int(D)$.

Proof. We prove first the assertion of the lemma under the assumption that t has the form $f(\mathbf{t})$, where f is an arithmetic function symbol, and $\mathbf{t}$ is a list $t_1, \dots, t_k$ of terms such that some term t_i is the variable V. Assume that f is definite; the case of indefinite f is similar. Then the antecedent of the implication to be proved is

$$\exists W \mathbf{J}(val_{\mathbf{t}}(\mathbf{J}) \wedge W = f(\mathbf{J})).$$

and

$$\exists J_i\, val_{t_i}(J_i) \;=\; \exists J_i\, val_V(J_i) \;=\; \exists J_i(V = J_i)$$

The last formula is equivalent to $\exists I(I = V)$, which is the consequent of the formula to be proved and is implied by $val_{\mathbf{t}}(\mathbf{J})$.

If t does not have the form described above then we reason by induction on the size of t. Assume, for instance, that t has the form $c(\mathbf{t})$, where c is a symbolic constant and $\mathbf{t}$ is a list $t_1, \dots, t_k$ of terms. (The case when the leading symbol of t is an arithmetic function symbol is similar.) The antecedent of the implication to be proved is

$$\exists W \mathbf{Y}(val_{\mathbf{t}}(\mathbf{Y}) \wedge W = (c\backslash k)(\mathbf{Y})), \tag{36}$$

where $\mathbf{Y}$ is a list $Y_1, \dots, Y_k$ of general variables of the same length as $\mathbf{t}$. Since V occurs in t, it occurs in some t_i. Then (36) implies $\exists Y_i\ val_{t_i}(Y_i)$, which by the induction hypothesis implies $\exists I(I = V)$. □

Lemma 12. *If B is a literal or comparison that occurs in N and does not contain indefinite function symbols, and V is a variable that occurs in B at least once in the scope of an arithmetic function symbol, then the formula*

$$\tau^B(B) \to \exists I(I = V)$$

is provable in $Int(D)$.

Proof. The proof is similar to the proof of Lemma 7 by Lifschitz (2021), but using Lemma 11 in place of Lemma 6. □

Lemma 13. *For any term t whose only basic symbols are numerals, and any variable V occurring in t, the formula*

$$\exists N\ val_t(N) \to \exists I(I = V)$$

is provable in $Int(D)$.

Proof. By induction on t. Since all basic symbols occurring in t are numerals, three cases are possible. *Case 1:* Term t is V. Then the antecedent $\exists N\ val_t(N)$ of the implication to be proven is $\exists N(N = V)$, which is equivalent to its consequent. *Case 2:* Term t has the form $f(t_1, \dots, t_k)$, where f is a definite arithmetic function symbol. Then the antecedent $\exists N\ val_t(N)$ of the implication to be proven is

$$\exists N \mathbf{J}(val_{\mathbf{t}}(\mathbf{J}) \wedge N = f(\mathbf{J})).$$

where $\mathbf{J}$ is a list of integer variables. Then, V occurs in some term t_i. The formula above implies $\exists I\ val_{t_i}(I)$. By the induction hypothesis, $\exists I(I = V)$ follows. *Case 3:* Term t has the form $f(t_1, \dots, t_k)$, where f is a indefinite arithmetic function symbol. Similar to Case 2. □

Lemma 14. *If B is a numeric equation of the form $t = f(\mathbf{t})$ such that f is an indefinite arithmetic function, and V is a variable that occurs in B, then formula*

$$\tau^B(B) \to \exists I(I = V)$$

is provable in $Int(D)$.

Proof. The antecedent of the formula to be proved is

$$\exists X_1 X_2(val_t(X_1) \wedge val_{f(\mathbf{t})}(X_2) \wedge X_1 = X_2),$$

which is equivalent to

$$\exists X(val_t(X) \wedge val_{f(\mathbf{t})}(X)). \tag{37}$$

Case 1: V occurs in t. Formula (37) can be expanded as

$$\exists X(val_t(X) \wedge \exists \mathbf{J} K(val_{\mathbf{t}}(\mathbf{J}) \wedge K = X \wedge F(\mathbf{J}, K))).$$

The above sentence implies $\exists K\ val_t(K)$. Then $\exists I(I = V)$ follows by Lemma 13, because all basic symbols occurring in t are numerals. *Case 2:* V occurs in $f(\mathbf{t})$. Formula (37) implies $\exists X\ val_{f(\mathbf{t})}(X)$, and hence $\exists I(I = V)$ follows by Lemma 11. □

Lemma 15. *If V is a variable that occurs in $Body$ at least once in the scope of an arithmetic function symbol, then*

$$\tau^B(Body) \to \exists I(I = V)$$

is provable in $Int(D)$.

Proof. The variable V occurs at least once in the scope of an arithmetic function symbol in some member of $Body$. The assertion of the lemma is given by Lemma 14 if that member is a comparison containing an indefinite function symbol, and from Lemma 12 otherwise. □

Lemma 16. *Let $\mathbf{t}$ be a list of terms occurring in N, let $V_1, \dots, V_k$ be all variables that occur in $\mathbf{t}$ in the scope of an arithmetic function symbol, let $\mathbf{Y}$ be a list of general variables, and let c be a symbolic constant. The implication*

$$\textstyle\bigwedge_{i=1}^{k} \exists I(I = V_i) \to \forall \mathbf{Y}(val_{\mathbf{t}}(\mathbf{Y}) \to (c/k)(\mathbf{Y})), \tag{38}$$

where I is a integer variable, is equivalent in $Int(D)$ to its consequent

$$\forall \mathbf{Y}(val_{\mathbf{t}}(\mathbf{Y}) \to (c/k)(\mathbf{Y})). \tag{39}$$

Proof. It is sufficient to show that, for every i,

$$\exists I(I = V_i) \to \forall \mathbf{Y}(val_{\mathbf{t}}(\mathbf{Y}) \to (c/k)(\mathbf{Y}))$$

is equivalent to (39). The last formula is equivalent to

$$\forall \mathbf{Y}(\exists I(I = V_i) \wedge val_{\mathbf{t}}(\mathbf{Y}) \to (c/k)(\mathbf{Y})).$$

By Lemma 11, the conjunction in the antecedent is equivalent to to its second conjunctive term. □

Proof of Theorem 2. We need to show that formulas (12) and

$$\tau^*(H \leftarrow B_1 \wedge \dots \wedge B_n) \tag{40}$$

are equivalent to each other in $Int(D)$. Let $Body$ stand for the conjunction $B_1 \wedge \dots \wedge B_n$ so that $\tau^B(Body)$ and $\nu(Body)$ respectively are

$$\tau^B(B_1) \wedge \dots \wedge \tau^B(B_n) \quad \text{and} \quad \nu(B_1) \wedge \dots \wedge \nu(B_n).$$

Let H be an atom $c(\mathbf{t})$, where $\mathbf{t}$ is a tuple $t_1, \dots, t_k$. (The case when H is empty is similar.) Formula (40) has the form

$$\widetilde{\forall}(val_{\mathbf{t}}(\mathbf{Y}) \wedge \tau^B(Body) \to (c/k)(\mathbf{Y})),$$

which is equivalent to

$$\widetilde{\forall}\left(\tau^B(Body) \to \forall \mathbf{Y}(val_{\mathbf{t}}(\mathbf{Y}) \to (c/k)(\mathbf{Y}))\right). \tag{41}$$

By Lemma 16, the consequent of (41) is equivalent to (38), where $V_1, \dots, V_k$ are all variables occurring in $\mathbf{t}$ in the

scope of some arithmetic function symbol. Then the variables $V_{k+1}, \dots, V_m$ occur in the scope of an arithmetic function symbol in $Body$. By Lemma 15, it follows that the formula

$$\tau^B(Body) \to \exists I(I = V_i)$$

is provable in $Int(D)$ for every i from $\{k+1, \dots, m\}$. Hence the antecedent $\tau^B(Body)$ of (41) is equivalent to

$$\bigwedge_{i=k+1}^{m} \exists I(I = V_i) \wedge \tau^B(Body),$$

and thus (41) is equivalent to

$$\widetilde{\forall}\left(\bigwedge_{i=1}^{m} \exists I(I = V_i) \wedge \tau^B(Body) \to F\right).$$

The last formula can be rewritten as

$$\widetilde{\forall}(C \to (\tau^B(Body) \to F)). \tag{42}$$

From Lemmas 6(i,iii,iv,v) and 10 we can conclude that formula (42) is equivalent to

$$\widetilde{\forall}(C \to (\nu(Body) \to \forall \mathbf{Y}(\mathbf{Y} = p2f(\mathbf{t}) \to (c/k)(\mathbf{Y})))),$$

which is equivalent to

$$\widetilde{\forall}(C \to (\nu(Body) \to \nu(H))).$$

The only part of the last formula that contains any of the variables V_i is C. Consequently that formula is equivalent to

$$\widetilde{\forall}\left(\bigwedge_i \exists V_i(I_i = V_i) \to (\nu(Body) \to \nu(H))\right). \tag{43}$$

Since the antecedent $\bigwedge_i \exists V_i(I_i = V_i)$ is provable in $Int(D)$, formula (43) is equivalent in $Int(D)$ to (12). □

9 Conclusion

This paper describes an approach to transforming mini-GRINGO rules into first-order sentences that involves a preprocessing step – converting the rule to a normal form. We expect that the new translation will be implemented in a future version of the proof assistant ANTHEM, and that it will make ANTHEM easier to use.

We plan to extend this translation method to answer set programming languages that include useful programming constructs not available in mini-GRINGO, such as conditional literals and aggregates.

Acknowledgements

We would like to thank Michael Gelfond, Zachary Hansen, Tobias Stolzmann and anonymous referees for their comments on earlier versions of this paper. This material is based upon work supported by the National Science Foundation under CAREER award 2338635, USA. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

AI Declaration

The authors have not employed any Generative AI tools.

References

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Maratea, M.; Ricca, F.; and Schaub, T. 2020. ASP-Core-2 input language format. *Theory and Practice of Logic Programming* 20:294–309.

Clark, K. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. New York: Plenum Press. 293–322.

Fandinno, J., and Lifschitz, V. 2023. 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*.

Fandinno, J.; Glinzer, C.; Hansen, Z.; Heuer, J.; Lierler, Y.; Lifschitz, V.; Schaub, T.; and Stolzmann, T. 2025. ANTHEM: answer set programming and automated theorem proving. *Theory and Practice of Logic Programming*. To appear.

Fandinno, J.; Lifschitz, V.; and Temple, N. 2024. Locally tight programs. *Theory and Practice of Logic Programming*.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2011. Stable models and circumscription. *Artificial Intelligence* 175:236–263.

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. Available at https://github.com/potassco/guide/releases/.

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. 2019. *Answer Set Programming*. Springer.

Lifschitz, V. 2021. Transforming gringo rules into formulas in a natural way. In *Proceedings of European Conference on Artificial Intelligence*.

Lifschitz, V. 2025. Generalizing the syntax of terms in minigringo. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*.

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.

Pearce, D., and Valverde, A. 2005. A first order nonmonotonic extension of constructive logic. *Studia Logica* 80:323–348.

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