Transforming Gringo Rules into Formulas in a Natural Way

Vladimir Lifschitz

University of Texas at Austin, USA
lifschitzv@gmail.com

Abstract. Research on the input language of the ASP grounder GRINGO uses a translation that converts rules in that language into first-order formulas. That translation often transforms short rules into formulas that are syntactically complex. In this note we identify a class of rules that can be transformed into formulas in a simpler, more natural way. The new translation contributes to our understanding of the relationship between the language of GRINGO and first-order languages.

1 Introduction

The semantics of some rules in the input language of the ASP grounder GRINGO [1, 2] can be characterized in terms of a translation into the language of first-order logic [3, Section 6]. The transformation τ^*, defined in that paper, produces formulas with two sorts—with "program variables" for arbitrary precomputed terms and "integer variables" for numerals. This transformation can be used, for instance, to characterize strong equivalence between GRINGO programs in terms of a similar condition on first-order formulas [3, Proposition 4]. It is used also in the design of the proof assistant ANTHEM [5].

The formulas produced by τ^* may be quite complicated, even in application to short rules, which makes it difficult to use them for reasoning about programs. For example, the result of applying τ^* to the rule

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

is

$$\forall X(\exists Z(Z = X \wedge p(Z)) \rightarrow \forall Z_1(\exists IJ(Z_1 = I + J \wedge I = X \wedge J = \overline{1}) \rightarrow q(Z_1))). \tag{2}$$

(In formulas, we use X, Y, Z as program variables, and I, J, K, L, M, N as integer variables; $\overline{1}$ is the numeral representing the number 1.)

Fortunately, complicated formulas produced by τ^* are often equivalent to much simpler formulas. In this note, equivalence of formulas is understood as equivalence in intuitionistic logic (see Section 3 for details). The use of intuitionistically acceptable simplifications in this context is essential because such simplifications do not affect the class of stable models [4]. For example, formula (2) is equivalent to

$$\forall X(p(X) \rightarrow \forall Z_1 IJ(Z_1 = I + J \wedge I = X \wedge J = \overline{1} \rightarrow q(Z_1))).$$

It can be further rewritten as

$$\forall X(p(X) \to \forall IJ(I = X \wedge J = \overline{1} \to q(I + J)))$$

and then as

$$\forall I(p(I) \to q(I + \overline{1})). \tag{3}$$

Formula (3) is not only short but also natural as a representation of rule (1), in the sense that its syntactic form is similar to the syntactic form of the rule. If our intention is to study properties of a program containing rule (1) using a representation of this rule in a first-order language, then representing it by a simple formula, such as (3), instead of (2) will make our work easier. This is particularly important if we plan to reason "manually," without the assistance of automated reasoning tools.

The goal of this paper is to identify a subset of the domain of τ^* for which transforming a rule into a formula can be performed "in a natural way." The new translation ν produces, whenever it is defined, a sentence equivalent to the result of applying τ^*. For example, in application to rule (1) the translation ν gives formula (3).

In the next section we describe a class of rules that can be transformed into formulas in a natural way. After a review of the syntax of formulas in Section 3, the translation ν is defined in Sections 4 and 5, and its equivalence to τ^* is proved in Sections 6 and 7. The note is concluded by a discussion of the analogy between regular rules and first-order formulas.

2 Regular Rules

The translation ν is defined on the class of rules that we call "regular."

Recall that in the definition of the syntax of rules, *program terms*, or *p-terms* for short, are defined as expressions formed from numerals, symbolic constants, program variables, and the symbols *inf* and *sup* using the binary function symbols

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

[3, Section 2]. (We call them p-terms to distinguish them from "f-terms" that are allowed in formulas; see Section 3 below.) About a p-term we say that it is *a regular term of the first kind* if

- it contains no function symbols other than $+$, $-$, $\times$, and
- symbolic constants and the symbols *inf*, *sup* do not occur in it in the scope of function symbols.

A regular term of the second kind is a p-term of the form $t_1 \mathinner{..} t_2$, where t_1 and t_2 are regular terms of the first kind that contain neither symbolic constants nor the symbols *inf*, *sup*.

Rules are defined as expressions of the form

$$H \leftarrow B_1 \wedge \cdots \wedge B_n \tag{4}$$

($n \geq 0$), where

– the head H is either an atom (then (4) is a *basic rule*), or an atom in braces (then (4) is a *choice rule*), or empty (then (4) is a *constraint*), and
– each member B_i of the body is a literal or a comparison

[3, Section 2]; see that paper for a complete definition of the syntax of rules. We say that a rule (4) is *regular* if it satisfies the following conditions:

1. Every p-term occurring in it is regular (of the first or second kind).
2. If B_i is a literal then it does not contain terms of the second kind.
3. If B_i is a comparison that contains a term of the second kind then B_i has the form $t_1 = t_2 \mathrel{..} t_3$, where t_1 is a term of the first kind different from symbolic constants and from the symbols *inf*, *sup*.

Condition 1 eliminates, for instance, rules containing any of the terms

$$X/Y,\ \overline{5} \times (X \mathrel{..} Y),\ \mathit{london} + \overline{5},\ \mathit{london} \mathrel{..} \overline{5}.$$

Condition 2 eliminates, for instance, rules containing the atom $p(\overline{1} \mathrel{..} \overline{5})$ in the body. Condition 3 eliminates, for instance, rules containing any of the comparisons

$$X < \overline{1} \mathrel{..} \overline{5},\ X \mathrel{..} Y = \overline{1} \mathrel{..} \overline{5},\ \mathit{london} = \overline{1} \mathrel{..} X.$$

Some of these "irregular" constructs exemplify differences between the language of GRINGO and conventional mathematical notation. In a mathematical formula, for instance, the arguments of + are expected to represent objects for which addition has been defined. But including $\mathit{london}+\overline{5}$ in a GRINGO program is not considered an error (although the output of GRINGO will include the informational message `info: operation undefined`). The expression $X \mathrel{..} Y = \overline{1} \mathrel{..} \overline{5}$ in the body of a GRINGO rule expresses that the interval $\{X, \dots, Y\}$ contains at least one number between 1 and 5; a mathematician would not use the equal sign to say that two sets have a common element.

The comparison $X = \overline{1} \mathrel{..} \overline{5}$, which is allowed in the body a regular rule, expresses that the value of X is one of the numbers $1, \dots, 5$, and this use of the equal sign does not look natural either. We will return to this example in Section 8.

3 F-terms and Formulas

The language of f-terms and formulas is a two-sorted first-order language, with program variables (the same that occur in rules, see Section 2) and integer variables [5, Section 3]. The second sort is a subsort of the first. The signature of the language consists of

– numerals, symbolic constants and the symbols *inf*, *sup* 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*;

– pairs p/n, where p is a symbolic constant and n is a nonnegative integer, as n-ary predicate constants, and the comparison symbols (the same that occur in comparisons) as predicate binary predicate constants.

An atomic formula $(p/n)(t_1, \ldots, t_n)$ can be abbreviated as $p(t_1, \ldots, t_n)$. An atomic formula $\prec (t_1, t_2)$, where $\prec$ is a comparison symbol, can be written as $t_1 \prec t_2$. *Formulas* are formed from atomic formulas using the propositional connectives

$$\bot \text{ (“false”)}, \ \wedge, \ \vee, \ \rightarrow$$

and the quantifiers $\forall$, $\exists$ as usual in first-order languages.

We use $\top$ as shorthand for $\bot \rightarrow \bot$, $\neg F$ as shorthand for $F \rightarrow \bot$, and $F \leftrightarrow G$ as shorthand for $(F \rightarrow G) \wedge (G \rightarrow F)$.

By *Int* we denote the formal system of intuitionistic logic with equality for the language described above. The natural deduction version of *Int* can be obtained from the standard natural deduction formulation of classical first-order logic [6, Sections 1.2.1, 1.2.2] by removing the law of the excluded middle from the list of axioms. The $\forall$-elimination rule of *Int* sanctions eliminating a universal quantifier that binds a program variable by substituting an f-term of either sort. When a quantifier binding an integer variable is eliminated, the f-term substituted for it is required to be of the sort *integer*. The $\exists$-introduction rule is similar. For instance, the formula $\exists X(I = X)$ can be proved in *Int* by applying $\exists$-introduction to $I = I$, but the formula $\exists I(I = X)$ is not provable. (This formula expresses that the value of X is a numeral.)

We say that formulas F and G are *equivalent* to each other if the formula $F \leftrightarrow G$ is provable in *Int*.

4 Natural Translation, Part 1

According to Condition 3 in the definition of a regular rule (Section 2), the left-hand side of a comparison in such a rule is a regular term of the first kind. If the right-hand side is of the first kind as well then we say that the comparison is *of the first kind*; otherwise it has the form $t_1 = t_2 \mathinner{..} t_3$, and we call it a comparison *of the second kind.*

Applying the translation ν to a regular rule (4) involves substituting integer variables for the variables that occur in that rule at least once in the scope of a function symbol or in a comparison of the second kind. Make the list $X_1, \ldots, X_m$ of all such variables, and choose m distinct integer variables $I_1, \ldots, I_m$. For any tuple $\mathbf{t}$ of regular terms of the first kind that occur in (4), the result of substituting $I_1, \ldots, I_m$ for $X_1, \ldots, X_m$ in $\mathbf{t}$ is a tuple of f-terms. The operator that performs this substitution will be denoted by p2f (“p-terms to f-terms”). For instance, in the case of rule (1), p2f$(X + \overline{1})$ is $I + \overline{1}$.

Prior to defining the translation ν we will define the auxiliary transformation ν', which will be used to translate the head H and the members $B_1, \ldots, B_n$ of the body of the rule. The definition of ν' is particularly simple if we restrict attention to the case when the head of the rule does not contain terms of the second kind:

- If $\mathbf{t}$ is a tuple of regular terms of the first kind then
  - $\nu'(p(\mathbf{t}))$ is $p(\mathrm{p2f}(\mathbf{t}))$,
  - $\nu'(\mathit{not}\ p(\mathbf{t}))$ is $\neg p(\mathrm{p2f}(\mathbf{t}))$,
  - $\nu'(\mathit{not}\ \mathit{not}\ p(\mathbf{t}))$ is $\neg\neg p(\mathrm{p2f}(\mathbf{t}))$,
  - $\nu'(\{p(\mathbf{t})\})$ is $p(\mathrm{p2f}(\mathbf{t})) \vee \neg p(\mathrm{p2f}(\mathbf{t}))$.
- The result of applying ν' to the empty string is $\bot$.
- If $t_1 \prec t_2$ is a comparison of the first kind then $\nu'(t_1 \prec t_2)$ is $\mathrm{p2f}(t_1) \prec \mathrm{p2f}(t_2)$.
- $\nu'(t_1 = t_2 \mathinner{..} t_3)$ is $\mathrm{p2f}(t_2) \leq \mathrm{p2f}(t_1) \leq \mathrm{p2f}(t_3)$.

(We use $t_1 \leq t_2 \leq t_3$ as shorthand for $t_1 \leq t_2 \wedge t_2 \leq t_3$.) This definition is extended to the general case in the next section.

The result of applying the translation ν to rule (4) is defined as the sentence

$$\widehat{\forall}(\nu'(B_1) \wedge \cdots \wedge \nu'(B_n) \to \nu'(H)). \tag{5}$$

(We write $\widehat{\forall}F$ for the universal closure of a formula F.)

One example of applying ν is given in the introduction: ν turns rule (1) into formula (3). If (4) is the rule

$$\leftarrow p(X, Y, Z) \wedge X < Y \wedge Y = \overline{1} \mathinner{..} Z$$

then the substitution p2f replaces Y, Z by I_1, I_2; the result of applying ν is

$$\forall X I_1 I_2 \neg(p(X, I_1, I_2) \wedge X < I_1 \wedge \overline{1} \leq I_1 \leq I_2).$$

5 Natural Translation, Part 2

Now we turn to the general case, when the head of rule (4) can contain terms of the second kind. As in the previous section, we start by making the list $X_1, \ldots, X_m$ of variables that occur in the rule at least once in the scope of a function symbol or in a comparison of the second kind, and choose distinct integer variables $I_1, \ldots, I_m$. The result of applying ν' to an atom of the form

$$p(t_1 \mathinner{..} t'_1,\ t_2 \mathinner{..} t'_2,\ \ldots)$$

is the formula

$$\forall N_1 N_2 \cdots (\mathrm{p2f}(t_1) \leq N_1 \leq \mathrm{p2f}(t'_1) \wedge \mathrm{p2f}(t_2) \leq N_2 \leq \mathrm{p2f}(t'_2) \wedge \cdots \to p(N_1, N_2, \ldots)),$$

where $N_1, N_2, \ldots$ are distinct integer variables different from $I_1, \ldots, I_m$. For example, the translation ν turns the rule

$$q(\overline{1} \mathinner{..} X, \overline{1} \mathinner{..} Y) \leftarrow p(X, Y, Z)$$

into the formula

$$\forall I_1 I_2 Z (p(I_1, I_2, Z) \to \forall N_1 N_2 (\overline{1} \leq N_1 \leq I_1 \wedge \overline{1} \leq N_2 \leq I_2 \to q(N_1, N_2))).$$

To make ν' applicable to arbitrary atoms allowed in the head of a regular rule we should take into account the fact that such an atom can include arguments of both kinds, in any order. If $\mathbf{t}$ is the tuple

$$\mathbf{t}_1,\ t_1\,..\,t'_1,\ \mathbf{t}_2,\ \dots,\ \mathbf{t}_{k-1},\ t_{k-1}\,..\,t'_{k-1},\ \mathbf{t}_k,$$

where $k > 1$ and $\mathbf{t}_1, \dots, \mathbf{t}_k$ are tuples of regular terms of the first kind, then we define:

– $\nu'(p(\mathbf{t}))$ is

$$\begin{array}{l}\forall N_1 \cdots N_{k-1}(\bigwedge_{i=1}^{k-1}(\text{p2f}(t_i) \leq N_i \leq \text{p2f}(t'_i)) \;\rightarrow \\ \qquad p(\text{p2f}(\mathbf{t}_1), N_1, \text{p2f}(\mathbf{t}_2), \dots, \text{p2f}(\mathbf{t}_{k-1}), N_{k-1}, \text{p2f}(\mathbf{t}_k))),\end{array} \tag{6}$$

– $\nu'(\{p(\mathbf{t})\})$ is

$$\begin{array}{l}\forall N_1 \cdots N_{k-1}(\bigwedge_{i=1}^{k-1}(\text{p2f}(t_i) \leq N_i \leq \text{p2f}(t'_i)) \;\rightarrow \\ \qquad p(\text{p2f}(\mathbf{t}_1), N_1, \text{p2f}(\mathbf{t}_2), \dots, \text{p2f}(\mathbf{t}_{k-1}), N_{k-1}, \text{p2f}(\mathbf{t}_k))) \vee \\ \qquad \neg p(\text{p2f}(\mathbf{t}_1), N_1, \text{p2f}(\mathbf{t}_2), \dots, \text{p2f}(\mathbf{t}_{k-1}), N_{k-1}, \text{p2f}(\mathbf{t}_k))).\end{array} \tag{7}$$

For example, if (4) is the rule

$$\{q(\overline{1}\,..\,X, Y)\} \leftarrow p(X, Y)$$

then the result of applying ν is

$$\forall IY(p(I, Y) \rightarrow \forall N(\overline{1} \leq N \leq I \rightarrow q(N, Y) \vee \neg q(N, Y))).$$

It is clear that every variable occurring in sentence (5) corresponding to rule (4) is either a program variable from (4) different from $X_1, \dots, X_m$, or one of the integer variables $I_1, \dots, I_m$, or one of the integer variables N_i in the consequent $\nu'(H)$ of (5).

Theorem *For any regular rule R, the formula $\nu(R)$ is equivalent to $\tau^*(R)$.*

6 Review: Definition of τ^*

We reproduce here the definition of τ^* [3, Section 6] referenced in the proof of the theorem in the next section. The definition makes use of the formulas $val_t(Z)$, where t is a term and Z is a variable that does not occur in t. The definition of $val_t(Z)$ is recursive and includes the following clauses:

– if t is a numeral, a symbolic constant, a program variable, *inf*, or *sup* then $val_t(Z)$ is $Z = t$;
– if t is $t_1 + t_2$ then $val_t(Z)$ is

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

and similarly for $t_1 - t_2$ and $t_1 \times t_2$;

– if t is $t_1\,..\,t_2$ then $val_t(Z)$ is

$$\exists IJK(val_{t_1}(I) \wedge val_{t_2}(J) \wedge I \leq K \leq J \wedge Z = K).$$

(The other clauses are not required for calculating $val_t(Z)$ when t is regular.) A conjunction of the form

$$val_{t_1}(Z_1) \wedge \cdots \wedge val_{t_k}(Z_k)$$

can be written as

$$val_{t_1,\ldots,t_k}(Z_1,\ldots,Z_k).$$

The auxiliary translation τ^B is defined as follows:

– $\tau^B(p(\mathbf{t}))$ is $\exists \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \wedge p(\mathbf{Z}))$, where $\mathbf{Z}$ is a tuple of distinct program variables that do not occur in $\mathbf{t}$;
– $\tau^B(not\,p(\mathbf{t}))$ is $\exists \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \wedge \neg p(\mathbf{Z}))$;
– $\tau^B(not\ not\,p(\mathbf{t}))$ is $\exists \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \wedge \neg\neg p(\mathbf{Z}))$;
– $\tau^B(t_1 \prec t_2)$ is $\exists Z_1 Z_2(val_{t_1,t_2}(Z_1,Z_2) \wedge Z_1 \prec Z_2)$.

Then the result of applying τ^* to rule (4) is defined as the formula

$$\widehat{\forall}(\tau^B(B_1) \wedge \cdots \wedge \tau^B(B_n) \to H^*), \tag{8}$$

where H^* stands for

$$\begin{array}{r} \forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z})), \text{ if } H \text{ is } p(\mathbf{t}); \\ \forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z}) \vee \neg p(\mathbf{Z})), \text{ if } H \text{ is } \{p(\mathbf{t})\}; \\ \bot, \text{ if } H \text{ is empty.} \end{array}$$

7 Proof of the Theorem

Consider a regular rule (4), and let C be the conjunction

$$I_1 = X_1 \wedge \cdots \wedge I_m = X_m,$$

where $X_1,\ldots,X_m,I_1,\ldots,I_m$ are as in the definition of p2f (Section 5). A conjunction of the form $t_1 = t'_1 \wedge \cdots \wedge t_m = t'_m$ can be also written as $(t_1,\ldots,t_m) = (t'_1,\ldots,t'_m)$.

Lemma 1. *For any tuple $\mathbf{t}$ of regular terms of the first kind that occur in rule (4), the formulas*

(i) $C \to \forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \leftrightarrow \mathbf{Z} = \mathrm{p2f}(\mathbf{t}))$,
(ii) $C \to (\nu'(p(\mathbf{t})) \leftrightarrow \forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z})))$,
(iii) $C \to (\nu'(p(\mathbf{t})) \leftrightarrow \tau^B(p(\mathbf{t})))$,
(iv) $C \to (\nu'(not\ p(\mathbf{t})) \leftrightarrow \tau^B(not\ p(\mathbf{t})))$,
(v) $C \to (\nu'(not\ not\ p(\mathbf{t})) \leftrightarrow \tau^B(not\ not\ p(\mathbf{t})))$

are provable in Int.

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

$$C \to \forall Z(val_t(Z) \leftrightarrow Z = \text{p2f}(t)). \tag{9}$$

The proof is by induction on t. *Case 1:* t is one of the variables X_k $(1 \leq k \leq m)$. Then the consequent of (9) is $\forall Z(Z = X_k \leftrightarrow Z = I_k)$, and the antecedent C contains the conjunctive term $X_k = I_k$. *Case 2:* t is a variable different from $X_1, \dots, X_m$, or a numeral, or a symbolic constant, or one of the symbols *inf*, *sup*. Then the consequent is $\forall Z(Z = t \leftrightarrow Z = t)$. *Case 3:* t contains a function symbol. Assume, for instance, that t is $t_1 + t_2$. Then p2f(t) is p2f(t_1)+p2f(t_2); this term and its subterms p2f(t_1), p2f(t_2) are of the sort *integer*. By the induction hypothesis, under the assumption C,

$$\begin{aligned} val_{t_1+t_2}(Z) &= \exists IJ(Z = I + J \wedge val_{t_1}(I) \wedge val_{t_2}(J)) \\ &\leftrightarrow \exists IJ(Z = I + J \wedge I = \text{p2f}(t_1) \wedge J = \text{p2f}(t_2)) \\ &\leftrightarrow \exists IJ(Z = \text{p2f}(t_1) + \text{p2f}(t_2) \wedge I = \text{p2f}(t_1) \wedge J = \text{p2f}(t_2)) \\ &\leftrightarrow Z = \text{p2f}(t) \wedge \exists I(I = \text{p2f}(t_1)) \wedge \exists J(J = \text{p2f}(t_2)). \end{aligned}$$

Since p2f(t_1) and p2f(t_2) are of the sort *integer*, the last two conjunctive terms are provable in *Int* and can be dropped.

(ii) By (i), under the assumption C,

$$\forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z})) \leftrightarrow \forall \mathbf{Z}(\mathbf{Z} = \text{p2f}(\mathbf{t}) \to p(\mathbf{Z})) \leftrightarrow p(\text{p2f}(\mathbf{t})) = \nu'(p(\mathbf{t})).$$

(iii) By (i), under the assumption C,

$$\tau^B(p(\mathbf{t})) = \exists \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \wedge p(\mathbf{Z})) \leftrightarrow \exists \mathbf{Z}(\mathbf{Z} = \text{p2f}(\mathbf{t}) \wedge p(\mathbf{Z})) \leftrightarrow p(\text{p2f}(\mathbf{t})) = \nu'(p(\mathbf{t})).$$

(iv), (v): Similar to (iii).

Lemma 2. *For any regular terms t_1, t_2, of the first kind that occur in rule (4), the formula*

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

is provable in Int.

Proof By Lemma 1(i), under the assumption C,

$$\begin{aligned} \tau^B(t_1 \prec t_2) &= \exists Z_1 Z_2(val_{t_1,t_2}(Z_1, Z_2) \wedge Z_1 \prec Z_2) \\ &\leftrightarrow \exists Z_1 Z_2(Z_1 = \text{p2f}(t_1) \wedge Z_2 = \text{p2f}(t_2) \wedge Z_1 \prec Z_2) \\ &\leftrightarrow \text{p2f}(t_1) \prec \text{p2f}(t_2) \\ &= \nu'(t_1 \prec t_2). \end{aligned}$$

Lemma 3. *For any regular term $t_1 \mathinner{..} t_2$ that occurs in rule (4), the formula*

$$C \to (val_{t_1 \mathinner{..} t_2}(Z) \leftrightarrow \exists K(\text{p2f}(t_1) \leq K \leq \text{p2f}(t_2) \wedge Z = K))$$

is provable in Int.

Proof Since $t_1 \mathrel{..} t_2$ is regular, t_1 and t_2 contain neither symbolic constants nor the symbols *inf*, *sup*. Since $t_1 \mathrel{..} t_2$ occurs in rule (4), all variables occurring in t_1, t_2 belong to the list $X_1, \dots, X_m$. It follows that the f-terms $\mathrm{p2f}(t_1)$ and $\mathrm{p2f}(t_2)$ are of the sort *integer*. By Lemma 1(i), under the assumption C,

$$\begin{aligned}
val_{t_1 \mathrel{..} t_2}(Z) &= \exists IJK(val_{t_1}(I) \wedge val_{t_2}(J) \wedge I \leq K \leq J \wedge Z = K)\\
&\leftrightarrow \exists IJK(I = \mathrm{p2f}(t_1) \wedge J = \mathrm{p2f}(t_2) \wedge I \leq K \leq J \wedge Z = K)\\
&\leftrightarrow \exists IJK(I = \mathrm{p2f}(t_1) \wedge J = \mathrm{p2f}(t_2) \wedge\\
&\qquad\qquad \mathrm{p2f}(t_1) \leq K \leq \mathrm{p2f}(t_2) \wedge Z = K)\\
&\leftrightarrow \exists I(I = \mathrm{p2f}(t_1)) \wedge \exists J(J = \mathrm{p2f}(t_2)) \wedge\\
&\qquad \exists K(\mathrm{p2f}(t_1) \leq K \leq \mathrm{p2f}(t_2) \wedge Z = K).
\end{aligned}$$

Since $\mathrm{p2f}(t_1)$ and $\mathrm{p2f}(t_2)$ are of the sort *integer*, the first two conjunctive terms are provable in *Int* and can be dropped.

Lemma 4. *If a comparison $t_1 = t_2 \mathrel{..} t_3$ occurs in rule (4) then the formula*

$$C \to (\nu'(t_1 = t_2 \mathrel{..} t_3) \leftrightarrow \tau^B(t_1 = t_2 \mathrel{..} t_3))$$

is provable in Int.

Proof By Lemma 1(i) and Lemma 3, under the assumption C,

$$\begin{aligned}
\tau^B(t_1 = t_2 \mathrel{..} t_3) &= \exists Z_1 Z_2(val_{t_1, t_2 \mathrel{..} t_3}(Z_1, Z_2) \wedge Z_1 = Z_2)\\
&\leftrightarrow \exists Z_1 Z_2(Z_1 = \mathrm{p2f}(t_1) \wedge val_{t_2 \mathrel{..} t_3}(Z_2) \wedge Z_1 = Z_2)\\
&\leftrightarrow val_{t_2 \mathrel{..} t_3}(\mathrm{p2f}(t_1))\\
&\leftrightarrow \exists K(\mathrm{p2f}(t_2) \leq K \leq \mathrm{p2f}(t_3) \wedge \mathrm{p2f}(t_1) = K)\\
&\leftrightarrow \mathrm{p2f}(t_2) \leq \mathrm{p2f}(t_1) \leq \mathrm{p2f}(t_3)\\
&= \nu'(t_1 = t_2 \mathrel{..} t_3).
\end{aligned}$$

Lemma 5. *For any tuple $\mathbf{t}$ of regular terms that occur in rule (4), the formulas*

(i) $C \to (\nu'(p(\mathbf{t})) \leftrightarrow \forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z})))$,
(ii) $C \to (\nu'(\{p(\mathbf{t})\}) \leftrightarrow \forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z}) \vee \neg p(\mathbf{Z}))$

are provable in Int.

Proof (i) If all members of the tuple $\mathbf{t}$ are of the first kind then the assertion holds by Lemma 1(ii). Otherwise, $\mathbf{t}$ can be represented in the form

$$\mathbf{t}_1,\ t_1 \mathrel{..} t'_1,\ \mathbf{t}_2,\ \dots,\ \mathbf{t}_{k-1},\ t_{k-1} \mathrel{..} t'_{k-1},\ \mathbf{t}_k,$$

where $k > 1$ and $\mathbf{t}_1, \dots, \mathbf{t}_k$ are tuples of terms of the first kind. Assume C; we need to derive the equivalence between $\nu'(p(\mathbf{t}))$ and

$$\forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z})).$$

The last formula can be written as

$$\begin{aligned}
\forall \mathbf{Z}_1 Z_1 \mathbf{Z}_2 \cdots \mathbf{Z}_{k-1} Z_{k-1} \mathbf{Z}_k(&val_{\mathbf{t}_1}(\mathbf{Z}_1) \wedge val_{t_1 \mathrel{..} t'_1}(Z_1) \wedge\\
&val_{\mathbf{t}_2}(\mathbf{Z}_2) \wedge \cdots \wedge val_{\mathbf{t}_{k-1}}(\mathbf{Z}_{k-1}) \wedge\\
&val_{t_{k-1} \mathrel{..} t'_{k-1}}(Z_{k-1}) \wedge val_{\mathbf{t}_k}(\mathbf{Z}_k)\\
&\qquad \to p(\mathbf{Z}_1, Z_1, \mathbf{Z}_2, \dots, \mathbf{Z}_{k-1}, Z_{k-1}, \mathbf{Z}_k)).
\end{aligned}$$

By Lemma 1(i), under the assumption C it is equivalent to

$$\begin{aligned}\forall \mathbf{Z}_1 Z_1 \mathbf{Z}_2 \cdots \mathbf{Z}_{k-1} Z_{k-1} \mathbf{Z}_k (\mathbf{Z}_1 = \text{p2f}(\mathbf{t}_1) \land val_{t_1\,..\,t'_1}(Z_1) \land \\ \mathbf{Z}_2 = \text{p2f}(\mathbf{t}_2) \land \cdots \land \mathbf{Z}_{k-1} = \text{p2f}(\mathbf{t}_{k-1}) \land \\ val_{t_{k-1}\,..\,t'_{k-1}}(Z_{k-1}) \land \mathbf{Z}_k = \text{p2f}(\mathbf{t}_k) \\ \rightarrow p(\mathbf{Z}_1, Z_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_{k-1}, Z_{k-1}, \mathbf{Z}_k))\end{aligned}$$

and can be further rewritten as

$$\begin{aligned}\forall Z_1 \cdots Z_{k-1}(val_{t_1\,..\,t'_1}(Z_1) \land \cdots \land val_{t_{k-1}\,..\,t'_{k-1}}(Z_{k-1}) \\ \rightarrow p(\text{p2f}(\mathbf{t}_1), Z_1, \text{p2f}(\mathbf{t}_2), \ldots, \text{p2f}(\mathbf{t}_{k-1}), Z_{k-1}, \text{p2f}(\mathbf{t}_k))).\end{aligned}$$

By Lemma 4, under the assumption C this formula is equivalent to

$$\begin{aligned}\forall Z_1 \cdots Z_{k-1}(\exists K(\text{p2f}(t_1) \leq K \leq \text{p2f}(t'_1) \land Z_1 = K) \land \cdots \land \\ \exists K(\text{p2f}(t_{k-1}) \leq K \leq \text{p2f}(t'_{k-1}) \land Z_{k-1} = K) \\ \rightarrow p(\text{p2f}(\mathbf{t}_1), Z_1, \text{p2f}(\mathbf{t}_2), \ldots, \text{p2f}(\mathbf{t}_{k-1}), Z_{k-1}, \text{p2f}(\mathbf{t}_k)))\end{aligned}$$

and can be further rewritten as

$$\begin{aligned}\forall Z_1 \cdots Z_{k-1} N_1 \cdots N_{k-1}(\text{p2f}(t_1) \leq N_1 \leq \text{p2f}(t'_1) \land Z_1 = N_1 \land \cdots \land \\ \text{p2f}(t_{k-1}) \leq N_{k-1} \leq \text{p2f}(t'_{k-1}) \land Z_{k-1} = N_{k-1} \\ \rightarrow p(\text{p2f}(\mathbf{t}_1), Z_1, \text{p2f}(\mathbf{t}_2), \ldots, \text{p2f}(\mathbf{t}_{k-1}), Z_{k-1}, \text{p2f}(\mathbf{t}_k))).\end{aligned}$$

This formula is equivalent to (6).

The proof of part (ii) is similar.

Lemma 6. *If a regular term t contains a function symbol then, for every variable X occurring in t, the formula*

$$\forall X(\exists Z\, val_t(Z) \rightarrow \exists I(I = X))$$

is provable in Int.

Proof By induction on t. Consider, for instance, the case when t has the form $t_1 + t_2$. Then the antecedent of the implication to be proved is

$$\exists ZIJ(Z = I + J \land val_{t_1}(I) \land val_{t_2}(J)).$$

Assume, for instance, that the part of t containing X is t_1. The formula above implies $\exists I\, val_{t_1}(I)$. If t_1 is X then the last formula is $\exists I(I = X)$, which is the consequent of the formula to be proved. Otherwise t_1 contains a function symbol, and $\exists I(I = X)$ follows by the induction hypothesis. If t is $t_1 - t_2$ $t_1 \times t_2$ or $t_1 \,..\, t_2$ then reasoning is similar.

Lemma 7. *If a conjunctive term B_i of the body of rule (4) is a literal or a comparison of the first kind then, for every variable X that occurs in B_i at least once in the scope of a function symbol, the formula $\tau^B(B_i) \rightarrow \exists I(I = X)$ is provable in Int.*

Proof *Case 1:* B_i is an atom $p(\mathbf{t})$. Then $\tau^B(B_i)$ is $\exists \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \wedge p(\mathbf{Z}))$, which implies $\exists \mathbf{Z}\, val_{\mathbf{t}}(\mathbf{Z})$ and further $\exists Z\, val_t(Z)$, where t is the component of the tuple $\mathbf{t}$ that contains X in the scope of a function symbol; $\exists I(I = X)$ follows by Lemma 6. *Case 2:* B_i is a literal of the form *not* $p(\mathbf{t})$ or *not not* $p(\mathbf{t})$. Similar to Case 1. *Case 3:* B_i is a comparison $t_1 \prec t_2$. Then $\tau^B(B_i)$ is

$$\exists Z_1 Z_2(val_{t_1,t_2}(Z_1, Z_2) \wedge Z_1 \prec Z_2),$$

which implies $\exists Z_j\, val_{t_j}(Z_j)$, where t_j is the part of the comparison that contains X in the scope of a function symbol; $\exists I(I = X)$ follows by Lemma 6.

Lemma 8. *For any regular term t and any variable X occurring in t, the formula*

$$\forall X(\exists N\, val_t(N) \to \exists I(I = X))$$

is provable in Int.

Proof By induction on t. *Case 1:* t is X. Then the antecedent $\exists N\, val_t(N)$ of the implication to be proved is $\exists N(N = X)$, which is equivalent to its consequent $\exists I(I = X)$. *Case 2:* t has the form $t_1 + t_2$, so that the antecedent is

$$\exists NIJ(N = I + J \wedge val_{t_1}(I) \wedge val_{t_2}(J)).$$

Assume, for instance, that the part of t containing X is t_1. The formula above implies $\exists I\, val_{t_1}(I)$. By the induction hypothesis, $\exists I(I = X)$ follows. *Case 3:* t has the form $t_1 - t_2$, $t_1 \times t_2$ or $t_1 \mathinner{..} t_2$. Similar to Case 2.

Lemma 9. *If a conjunctive term B_i of the body of rule (4) is a comparison of the second kind then, for every variable X that occurs in B_i, the formula $\tau^B(B_i) \to \exists I(I = X)$ is provable in Int.*

Proof The antecedent $\tau^B(B_i)$ of the formula to be proved is

$$\exists Z_1 Z_2(val_{t_1, t_2 \mathinner{..} t_3}(Z_1, Z_2) \wedge Z_1 = Z_2),$$

which is equivalent to

$$\exists Z\, val_{t_1, t_2 \mathinner{..} t_3}(Z, Z). \tag{10}$$

Case 1: X occurs in t_1. Formula (10) can be rewritten as

$$\exists Z(val_{t_1}(Z) \wedge \exists IJK(val_{t_2}(I) \wedge val_{t_3}(J) \wedge I \le K \le J \wedge Z = K)).$$

Consequently it implies $\exists K\, val_{t_1}(K)$, and $\exists I(I = X)$ follows by Lemma 8. *Case 2:* X occurs in $t_2 \mathinner{..} t_3$. Formula (10) implies $\exists Z\, val_{t_2 \mathinner{..} t_3}(Z)$, and $\exists I(I = X)$ follows by Lemma 6.

Proof of the Theorem We need to show that formulas (5) and (8) are equivalent to each other. Consider all variables from the set $\{X_1, \dots, X_m\}$ that occur in the head H of rule (4) in the scope of a function symbol. Let these variables be $X_1, \dots, X_k$; then each of the remaining variables $X_{k+1}, \dots, X_m$

occurs in the body of the rule in the scope of a function symbol or in a comparison of the second kind.

We will show first that the consequent H^* of (8) is equivalent to

$$\exists I(I = X_i) \to H^* \qquad (1 \le i \le k). \tag{11}$$

If H is an atom $p(\mathbf{t})$ then formula (11) is

$$\exists I(I = X_i) \to \forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z})),$$

and it is equivalent to

$$\forall \mathbf{Z}(\exists I(I = X_i) \land val_{\mathbf{t}}(\mathbf{Z}) \to p(\mathbf{Z}))). \tag{12}$$

Lemma 6 shows that the conjunction in the antecedent is equivalent to its second conjunctive term $val_{\mathbf{t}}(\mathbf{Z})$, so that formula (12) is equivalent to H^*. If H is $\{p(\mathbf{t})\}$ then reasoning is similar. If H is empty then $k = 0$, and there is nothing to prove.

It follows that H^* is equivalent to

$$\exists I(I = X_1) \to (\exists I(I = X_2) \to \dots (\exists I(I = X_k) \to H^*) \dots)$$

and consequently to

$$\bigwedge_{i=1}^{k} \exists I(I = X_i) \to H^*. \tag{13}$$

On the other hand, Lemmas 7 and 9 show that the formulas

$$\tau^B(B_1) \land \dots \land \tau^B(B_n) \to \exists I(I = X_i) \qquad (k+1 \le i \le m)$$

are provable in *Int*. It follows that the antecedent

$$\tau^B(B_1) \land \dots \land \tau^B(B_n)$$

of (8) is equivalent to

$$\bigwedge_{i=k+1}^{m} \exists I(I = X_i) \;\land\; \tau^B(B_1) \land \dots \land \tau^B(B_n). \tag{14}$$

From these observations about formulas (13) and (14) we can conclude that the result (8) of applying τ^* to rule (4) is equivalent to the formula

$$\widehat{\forall}\left(\bigwedge_{i=1}^{m} \exists I(I = X_i) \land \tau^B(B_1) \land \dots \land \tau^B(B_n) \to H^*\right),$$

which can be further rewritten as

$$\widehat{\forall}(C \to (\tau^B(B_1) \land \dots \land \tau^B(B_n) \to H^*)). \tag{15}$$

From Lemmas 1(iii,iv,v), 2, 4, 5 we can conclude that formula (15) is equivalent to

$$\widehat{\forall}(C \to (\nu'(B_1) \wedge \cdots \wedge \nu'(B_n) \to \nu'(H))).$$

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

$$\widehat{\forall}\left(\bigwedge_i \exists X_i(I_i = X_i) \to (\nu'(B_1) \wedge \cdots \wedge \nu'(B_n) \to \nu'(H))\right).$$

Since the antecedent $\bigwedge_i \exists X_i(I_i = X_i)$ is provable in *Int*, it can be dropped, which leads us to formula (5).

8 Discussion

It was observed long ago that the head and body of a rule are similar to the consequent and antecedent of an implication, and that choice expressions are similar to excluded middle formulas. For instance, the rule

$$\{q(X)\} \leftarrow p(X)$$

is similar to the formula

$$p(X) \to q(X) \vee \neg q(X).$$

The definition of the translation ν allows us to extend this analogy to regular rules containing arithmetic operations and comparisons:

1. A variable in a regular rule is similar to a variable for integers if it occurs at least once in the scope of a function symbol or in a comparison of the second kind. Otherwise it is similar to a variable for arbitrary precomputed terms.
2. The equal sign in a comparison of the second kind, such as $X = \overline{1}\,..\,\overline{5}$, is similar to the membership symbol: it expresses that the integer denoted by the left-hand side is an element of the set denoted by the right-hand side.
3. The atom in the head of a regular rule that contains the interval symbol, such as $q(\overline{1}\,..\,X, \overline{1}\,..\,Y)$, is similar to a universally quantified formula.

Acknowledgements

Thanks to Jorge Fandinno, Michael Gelfond, Yuliya Lierler, Torsten Schaub, and the anonymous referees for useful comments.

References

1. Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S.: Potassco User Guide. Available at https://github.com/potassco/guide/releases/ (2015)
2. Gebser, M., Harrison, A., Kaminski, R., Lifschitz, V., Schaub, T.: Abstract Gringo. Theory and Practice of Logic Programming **15** (2015) 449–463
3. 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)
4. Lifschitz, V., Pearce, D., Valverde, A.: A characterization of strong equivalence for logic programs with variables. In: Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2007) 188–200
5. Fandinno, J., Lifschitz, V., Lühne, P., Schaub, T.: Verifying tight programs with Anthem and Vampire. Theory and Practice of Logic Programming **20** (2020)
6. Lifschitz, V., Morgenstern, L., Plaisted, D.: Knowledge representation and classical logic. In van Harmelen, F., Lifschitz, V., Porter, B., eds.: Handbook of Knowledge Representation. Elsevier (2008) 3–88