TPLP **23** (4): 664–677, 2023.
doi:10.1017/S1471068423000224 First published online 18 July 2023

On Program Completion, with an Application to the Sum and Product Puzzle

VLADIMIR LIFSCHITZ
University of Texas, Austin, TX, USA
(*e-mail:* lifschitzv@gmail.com)

submitted 26 May 2023; revised 14 June 2023; accepted 15 June 2023;

Abstract

This paper describes a generalization of Clark's completion that is applicable to logic programs containing arithmetic operations and produces syntactically simple, natural looking formulas. If a set of first-order axioms is equivalent to the completion of a program, then we may be able to find standard models of these axioms by running an answer set solver. As an example, we apply this "reverse completion" procedure to the Sum and Product Puzzle.

KEYWORDS: answer set programming, program completion, stable models, sum and product puzzle, tight programs

1 Introduction

Program completion (Clark 1978; Lloyd and Topor 1984) is a transformation that converts logic programs into sets of first-order formulas. The study of completion improved our understanding of the relationship between these two knowledge representation formalisms, and it has been used in the design of answer set solvers (Lierler and Maratea 2004; Lin and Zhao 2004).

The definition of completion has been extended to programs with operations on integers (Fandinno *et al.* 2020). That generalized completion process produces formulas in a two-sorted first-order language (Lifschitz *et al.* 2019, Section 5). In addition to "general" variables, which range over both symbolic constants and (symbols for) integers, a formula in that language may include also variables ranging over integers only. The need to use a language with two sorts is explained by the fact that function symbols in a first-order language are supposed to represent total functions, and arithmetic operations are not defined on symbolic constants. In answer set programming languages, applying arithmetic operations to symbolic constants is usually handled in a different way; when a rule is instantiated, a substituition is not used unless it is "well formed" (Calimeri *et al.* 2020, Section 3).

In this paper, the idea of a natural translation (Lifschitz 2021) is used to define a version of generalized completion that is limited to relatively simple ("regular") rules but produces simpler, and more natural-looking, formulas. The modified completion operator is denoted by NCOMP, for "natural completion." For example, the natural completion of the one-rule program:

```
even(2*X) :- X = -10..10.
```
(1)

in the input language of the answer set solver CLINGO (Gebser *et al.* 2019) is the sentence:

$$\forall V(even(V) \leftrightarrow \exists I(\overline{-10} \leq I \leq \overline{10} \wedge V = \overline{2} * I)). \tag{2}$$

Here, V is a general variable, I is an integer variable, and $\overline{-10}$, $\overline{10}$, $\overline{2}$ are "numerals" – object constants representing integers.

Two theorems, stated in Section 3.4 and proved in Section 5, relate stable models of a regular program to standard models of its completion (standard in the sense that they interpret symbols related to integers as usual in arithmetic). These theorems extend well-known results due to (Fages 1991).

If a set of first-order axioms happens to be equivalent to the completion of a regular program, then we may be able to find standard models of these axioms by running an answer set solver. As an example, we apply this "reverse completion" procedure to a formalization of the Sum and Product Puzzle (https://en.wikipedia.org/wiki/Sum_and_Product_Puzzle). From the perspective of knowledge representation and automated reasoning, that puzzle presents a challenge: express it in a formal declarative language so that the answer can be found, or at least verified, by an automated reasoning tool. This has been accomplished using first-order axioms for Kripke-style possible worlds and the first-order theorem prover FOL (McCarthy 1990), and also using a modal logic of public announcements and the epistemic model checker DEMO (van Ditmarsch *et al.* 2005). More recently, Jayadev Misra proposed a simple first-order formalization that does not refer to possible worlds (Misra 2022, Section 2.8.3). In Section 4, we show that the answer to the puzzle can be found by applying the reverse completion process to a variant of his axiom set and then running CLINGO.

2 Review: Rules and formulas

2.1 Regular rules

To simplify presentation, we do not include here some of the programming constructs that are classified as regular in the previous publication on natural translations (Lifschitz 2021). As in the Abstract Gringo article (Gebser *et al.* 2015), rules will be written in abstract notation, which disregards some details related to representing rules by strings of ASCII characters. For example, rule (1) will be written as:

$$even(\overline{2} \times X) \leftarrow X = \overline{-10}\,..\,\overline{10}. \tag{3}$$

We assume that three disjoint countably infinite sets of symbols are selected: *numerals*, *symbolic constants*, and *(general) variables*. We assume that a 1-1 correspondence between numerals and integers is chosen; the numeral corresponding to an integer n is denoted by $\overline{n}$. *Precomputed terms* are numerals and symbolic constants. We assume that a total order on the set of precomputed terms is chosen so that, for all integers m and n,

- $\overline{m} < \overline{n}$ iff $m < n$, and
- every precomputed term t such that $\overline{m} < t < \overline{n}$ is a numeral.

https://doi.org/10.1017/S1471068423000224 Published online by Cambridge University Press

Regular terms are formed from numerals and variables using the binary function symbols $+$, $-$, $\times$. A *regular atom* is an expression of the form $p(\mathbf{t})$, where p is a symbolic constant and $\mathbf{t}$ is a tuple of symbolic constants and regular terms, separated by commas. *Regular comparisons* are expressions of the forms:

- $t_1 \prec t_2$, where each of t_1 and t_2 is a symbolic constant or a regular term, and $\prec$ is one of the comparison symbols $=$, $\neq$, $<$, $>$, $\leq$, $\geq$, and
- $t_1 = t_2 \mathrel{..} t_3$, where t_1, t_2, and t_3 are regular terms.

A *regular rule* is an expression of the form:

$$\mathit{Head} \leftarrow \mathit{Body}, \tag{4}$$

where

- *Head* is either a regular atom (then (4) is a *basic rule*), or a regular atom in braces (then (4) is a *choice rule*), or empty (then (4) is a *constraint*), and
- *Body* is a conjunction, possibly empty, of (i) regular atoms, possibly preceded by *not*, and (ii) regular comparisons.

For example, (3) is a regular rule.

A *regular program* is a finite set of regular rules. This is a special case of Abstract Gringo programs (Gebser *et al.* 2015), and stable models of a regular program are understood in the sense of the semantic of Abstract Gringo. Thus, stable models are sets of ground atoms that do not contain arithmetic operations.

2.2 Two-sorted formulas

A *predicate symbol* is a pair p/n, where p is a symbolic constant and n is a nonnegative integer. About a predicate symbol p/n we say that it *occurs* in a regular program Π if some atom of the form $p(t_1, \dots, t_n)$ occurs in one of the rules of Π.

For any regular program Π, by σ_Π we denote the two-sorted signature with the sort *general* and its subsort *integer*, which includes

- every numeral as an object constant of the sort *integer*,
- every symbolic constant as an object constant of the sort *general*,
- the symbols $+$, $-$, $\times$ as binary function constants with the argument sorts *integer* and the value sort *integer*,
- every predicate symbol p/n that occurs in Π as an n-ary predicate constant with the argument sorts *general*,
- the symbols $\neq$, $<$, $>$, $\leq$, $\geq$ as binary predicate constants with the argument sorts *general*.

A formula over σ_Π that has the form $(p/n)(\mathbf{t})$ can be abbreviated as $p(\mathbf{t})$. This convention allows us to view regular atoms occurring in Π as atomic formulas over σ_Π.

Conjunctions of equalities and inequalities can be abbreviated as usual in algebra; for instance, $t_1 \leq t_2 \leq t_3$ stands for $t_1 \leq t_2 \wedge t_2 \leq t_3$. An equality between tuples of terms $(t_1, \dots, t_k) = (t'_1, \dots, t'_k)$ is understood as the conjunction $t_1 = t'_1 \wedge \cdots \wedge t_k = t'_k$.

In this paper, integer variables are denoted by capital letters from the middle of the alphabet $(I, \dots, N)$, and general variables by letters from the end $(U, \dots, Z)$.

https://doi.org/10.1017/S1471068423000224 Published online by Cambridge University Press

3 Completion

3.1 Replacing variables

In the process of constructing the natural completion of a regular program Π, the bodies of rules of Π will be transformed into formulas over σ_Π. Since general variables are not allowed in a formula in the scope of an arithmetic operation, this process has to involve replacing some of them by integer variables.

A *critical variable* of a regular rule R is a general variable X such that at least one occurrence of X in R is in the scope of an arithmetic operation or is part of a comparison of the form $t_1 = t_2 \,..\, t_3$. For every regular rule R, choose a function f_R that maps its critical variables to pairwise distinct integer variables. This function f_R is extended to other subexpressions of R as follows. For a tuple $\mathbf{t}$ of symbolic constants and regular terms, $f_R(\mathbf{t})$ is the tuple of terms over σ_Π obtained from $\mathbf{t}$ by replacing all occurrences of every critical variable X with the integer variable $f_R(X)$. Applying f_R to a regular atom and to a comparison that does not contain intervals is defined in a similar way. The result of applying f_R to *not* A is defined as the formula $\neg f_R(A)$, and the result of applying f_R to a comparison $t_1 = t_2 \,..\, t_3$ is $f_R(t_2) \leq f_R(t_1) \leq f_R(t_3)$. Finally, applying f_R to the body $B_1 \wedge B_2 \wedge \cdots$ of R gives the formula $f_R(B_1) \wedge f_R(B_2) \wedge \cdots$.

For instance, if R is rule (3) then the variable X is critical, and f_R maps X to some integer variable I. It transforms the term $\overline{2} \times X$ in the head into $\overline{2} \times I$, and the body $X = \overline{-10} \,..\, \overline{10}$ into $\overline{-10} \leq I \leq \overline{10}$.

3.2 Completed definitions

Consider a regular program Π and a predicate symbol p/n that occurs in Π. The *definition* of p/n in Π is the set of all rules of Π that have the form:

$$p(\mathbf{t}) \leftarrow \mathit{Body}, \tag{5}$$

or

$$\{p(\mathbf{t})\} \leftarrow \mathit{Body}, \tag{6}$$

such that the length of the tuple $\mathbf{t}$ is n. The *completed definition* of p/n in Π is the sentence over σ_Π constructed as follows. Choose a tuple $\mathbf{V}$ of n general variables that do not occur in Π. For every rule R in the definition D of p/n in Π, by F_R we denote the formula:

$$f_R(\mathit{Body}) \wedge \mathbf{V} = f_R(\mathbf{t}),$$

if R is (5), and

$$f_R(\mathit{Body}) \wedge \mathbf{V} = f_R(\mathbf{t}) \wedge p(\mathbf{V}),$$

if R is (6). The completed definition of p/n in Π is the sentence:

$$\forall \mathbf{V} \left(p(\mathbf{V}) \leftrightarrow \bigvee_{R \in D} \exists \mathbf{U}_R F_R \right), \tag{7}$$

where $\mathbf{U}_R$ is the list of all variables occurring in $f_R(\mathit{Body})$ or in $f_R(\mathbf{t})$.

For example, if the only rule R of the program is (3), and p/n is $\mathit{even}/1$, then F_R is

$$\overline{-10} \leq I \leq \overline{10} \;\wedge\; V = \overline{2} * I,$$

where I is $f_R(X)$. The completed definition of $\mathit{even}/1$ is (2).

https://doi.org/10.1017/S1471068423000224 Published online by Cambridge University Press

The formula obtained from the completed definition (7) by replacing the global variables $\mathbf{V}$ with fresh integer variables will be called the *arithmetic completed definition* of p/n in Π. For example, the arithmetic completed definition of $even/1$ in program (3) is

$$\forall N(even(N) \leftrightarrow \exists I(\overline{-10} \leq I \leq \overline{10} \wedge N = \overline{2} * I)). \tag{8}$$

The arithmetic completed definition is entailed by the completed definition, but not the other way around. For example, from formula (2) we can derive $\neg even(t)$ for every symbolic constant t, but this conclusion is not warranted by formula (8).

3.3 Natural completion

The *natural completion* NCOMP(Π) of a regular program Π is the set of sentences that includes

- for every predicate symbol p/n occurring in Π, its completed definition in Π, and
- for every constraint $\leftarrow Body$ in Π, the universal closure of the formula:

$$\neg f_{\leftarrow Body}(Body).$$

Consider, for example, the program that consists of rule (3), the choice rule:

$$\{foo(X)\} \leftarrow even(X), \tag{9}$$

and the constraint:

$$\leftarrow not\ foo(\overline{0}). \tag{10}$$

Its natural completion consists of the completed definition (2) of $even/2$, the completed definition of $foo/1$:

$$\forall V(foo(V) \leftrightarrow \exists X(even(X) \wedge X = V \wedge foo(V))),$$

which can be rewritten[1] as:

$$\forall V(foo(V) \rightarrow even(V)),$$

and the sentence $\neg\neg foo(\overline{0})$, which is equivalent to $foo(\overline{0})$.

3.4 Relation to stable models

The *Herbrand base* of a regular program Π is the set of all regular atoms $p(t_1, \ldots, t_n)$ such that p/n occurs in Π and $t_1, \ldots, t_n$ are precomputed terms. If S is a subset of the Herbrand base of Π then $S^{\uparrow}$ is the interpretation of the signature σ_{Π} defined as follows:

(i) the universe of the sort *general* in $S^{\uparrow}$ is the set of all precomputed terms;
(ii) the universe of the sort *integer* in $S^{\uparrow}$ is the set of all numerals;
(iii) for every precomputed term t, $S^{\uparrow}(t) = t$;
(iv) for every pair m, n of integers, $S^{\uparrow}(\overline{m} + \overline{n}) = \overline{m+n}$, and similarly for subtraction and multiplication;

[1] When we talk about equivalent transformations of a completed definition, equivalence is understood in the sense of classical first-order logic.

https://doi.org/10.1017/S1471068423000224 Published online by Cambridge University Press

(v) for every pair t_1, t_2 of precomputed terms, $S^{\uparrow}$ satisfies $t_1 < t_2$ iff the relation $<$ holds for the pair t_1, t_2, and similarly for the other comparison symbols.

Theorem 1
For any regular program Π and any subset S of its Herbrand base, if S is a stable model of Π then $S^{\uparrow}$ satisfies NCOMP(Π).

The *positive predicate dependency graph* of a regular program Π is the directed graph defined as follows. Its vertices are the predicate symbols p/n occurring in Π. It has an edge from p/n to q/m if Π has a rule (4) such that

- *Head* has the form $p(t_1, \dots, t_n)$ or $\{p(t_1, \dots, t_n)\}$, and
- one of the conjunctive terms of *Body* has the form $q(t_1, \dots, t_m)$.

A regular program Π is *tight* if its positive predicate dependency graph is acyclic.

For example, the positive predicate dependency graph of program (3), (9), and (10) has one edge, from *foo*/1 to *even*/1. This program is tight.

Theorem 2
For any tight regular program Π and any subset S of its Herbrand base, S is a stable model of Π iff $S^{\uparrow}$ satisfies NCOMP(Π).

4 The puzzle

Two mathematicians, S and P, talk about two integers, M and N. S knows the sum $M + N$, and P knows the product $M \times N$. Both S and P know also that the integers are greater than 1; that their sum is not greater than 100; and that N is greater than M. The following conversation occurs:

1. *S says: P does not know M and N.*
2. *P says: Now I know M and N.*
3. *S says: Now I also know M and N.*

What are M and N?

4.1 First-order axioms

Jayadev Misra's approach to translating this puzzle into a first-order language (Misra 2022, Section 2.8.3) involves the use of binary predicates $b_0, \dots, b_3$. The formula $b_0(M, N)$ expresses that before the beginning of the conversation, the pair M, N was considered a possible solution. This can be expressed by the formula:

$$b_0(M, N) \leftrightarrow 1 < M < N \wedge M + N \leq 100. \tag{11}$$

The formula $b_1(M, N)$ expresses that M, N was considered a possible solution at step 1, that is, after hearing the words "P does not know M and N," and so forth.

There are several ways to write axioms for b_1, b_2, and b_3. One possibility is described below.

We say that an integer I is *puzzling at time 0* if is there is more than one way to represent it as the product of two numbers J, K satisfying $b_0(J, K)$:

https://doi.org/10.1017/S1471068423000224 Published online by Cambridge University Press

$$\begin{aligned}puzzling_0(I) \leftrightarrow \exists J_1 K_1 J_2 K_2 (b_0(J_1, K_1) \wedge b_0(J_2, K_2) \\ \wedge I = J_1 \times K_1 = J_2 \times K_2 \wedge J_1 \neq J_2).\end{aligned} \tag{12}$$

We say that an integer I is *possibly easy* if it can be represented as the sum of two numbers J and K satisfying $b_0(J, K)$ such that $J \times K$ is not puzzling at time 0:

$$possibly\_easy(I) \leftrightarrow \exists JK(b_0(J, K) \wedge I = J + K \wedge \neg puzzling_0(J \times K)). \tag{13}$$

Then the assumption:

"at Step 1, S knows that P does not know M and N"

can be expressed by the axiom:

$$b_1(M, N) \leftrightarrow b_0(M, N) \wedge \neg possibly\_easy(M + N). \tag{14}$$

We say that an integer I is *puzzling at time 1* is there is more than one way to represent it as the product of two numbers J, K satisfying $b_1(J, K)$:

$$\begin{aligned}puzzling_1(I) \leftrightarrow \exists J_1 K_1 J_2 K_2 (b_1(J_1, K_1) \wedge b_1(J_2, K_2) \\ \wedge I = J_1 \times K_1 = J_2 \times K_2 \wedge J_1 \neq J_2).\end{aligned} \tag{15}$$

The assumption

"at Step 2, P knows M and N",

can be expressed by the axiom

$$b_2(M, N) \leftrightarrow b_1(M, N) \wedge \neg puzzling_1(M \times N). \tag{16}$$

We say that an integer I is *puzzling at time 2* is there is more than one way to represent it as the sum of two numbers J, K satisfying $b_2(J, K)$:

$$\begin{aligned}puzzling_2(I) \leftrightarrow \exists J_1 K_1 J_2 K_2 (b_2(J_1, K_1) \wedge b_2(J_2, K_2) \\ \wedge I = J_1 + K_1 = J_2 + K_2 \wedge J_1 \neq J_2).\end{aligned} \tag{17}$$

Finally, the assumption

"at Step 3, S knows M and N",

can be expressed by the axiom:

$$b_3(M, N) \leftrightarrow b_2(M, N) \wedge \neg puzzling_2(M + N). \tag{18}$$

Since axioms (11)–(18) form a chain of explicit definitions, the predicates represented by the symbols:

$$b_0/2,\ \ldots,\ b_3/2,\ puzzling_0/1,\ \ldots,\ puzzling_2/1,\ possibly\_easy/1, \tag{19}$$

are uniquely defined, assuming that variables range over the integers and that the symbols:

$$+ \quad \times \quad < \quad \leq,$$

are interpreted in the standard way. To solve the Sum and Product Puzzle, we will calculate the extents of these predicates.

This will be accomplished by running CLINGO on the "reverse completion" of axioms (11)–(18).

4.2 Reverse completion

Consider the regular program

$$\begin{array}{rcl}
b_0(XM, XN) & \leftarrow & \overline{1} < XM \wedge XM < XN \wedge XM + XN \leq \overline{100}, \\
puzzling_0(XI) & \leftarrow & b_0(XJ_1, XK_1) \wedge b_0(XJ_2, XK_2) \wedge XI = XJ_1 \times XK_1 \\
 & & \quad \wedge XJ_1 \times XK_1 = XJ_2 \times XK_2 \wedge XJ_1 \neq XJ_2, \\
possibly_easy(XI) & \leftarrow & b_0(XJ, XK) \wedge XI = XJ + XK \\
 & & \wedge \; not \; puzzling_0(XJ \times XK), \\
b_1(XM, XN) & \leftarrow & b_0(XM, XN) \wedge not \; possibly_easy(XM + XN), \\
puzzling_1(XI) & \leftarrow & b_1(XJ_1, XK_1) \wedge b_1(XJ_2, XK_2) \wedge XI = XJ_1 \times XK_1 \\
 & & \quad \wedge XJ_1 \times XK_1 = XJ_2 \times XK_2 \wedge XJ_1 \neq XJ_2, \\
b_2(XM, XN) & \leftarrow & b_1(XM, XN) \wedge not \; puzzling_1(XM \times XN), \\
puzzling_2(XI) & \leftarrow & b_2(XJ_1, XK_1) \wedge b_2(XJ_2, XK_2) \wedge XI = XJ_1 + XK_1 \\
 & & \quad \wedge XJ_1 + XK_1 = XJ_2 + XK_2 \wedge XJ_1 \neq XJ_2, \\
b_3(XM, XN) & \leftarrow & b_2(XM, XN) \wedge not \; puzzling_2(XM + XN).
\end{array} \tag{20}$$

These rules are obtained from equivalences (11)–(18) by:

- replacing the equivalence signs $\leftrightarrow$ by left arrows,
- dropping existential quantifiers,
- replacing integer variables by general variables, and
- replacing $\neg$ by *not*.

The natural completion of program (20) looks very similar to axiom set (11)–(18). There is a difference though: the former consists of formulas over the two-sorted signature described in Section 2.2, and the axioms formalizing the Sum and Product puzzle are one-sorted; there are no general variables in them. Consider then the *arithmetic* completed definitions of predicate symbols (19) (see Section 3.2). Those are one-sorted formulas, and they are equivalent to the universal closures of the corresponding axioms. For example, the completed definition of the predicate symbol $b_0/2$ in program (20) is

$$\begin{array}{l}
\forall XM \; XN(b_0(XM, XN) \leftrightarrow \\
\qquad \exists MN(\overline{1} < M \wedge M < N \wedge M + N \leq \overline{100} \wedge XM = M \wedge XN = N)),
\end{array}$$

and the arithmetic completed definition of this symbol is the one-sorted formula:

$$\begin{array}{l}
\forall M_1 N_1(b_0(M_1, N_1) \leftrightarrow \\
\qquad \exists MN(\overline{1} < M \wedge M < N \wedge M + N \leq \overline{100} \wedge M_1 = M \wedge N_1 = N)).
\end{array}$$

This formula is equivalent to the universal closure of axiom (11). Similarly, the arithmetic completed definition of $puzzling_0/2$ is equivalent to the universal closure of axiom (12), and so forth.

4.3 Calculating the answer

By running CLINGO, we can determine that program (20) has a unique stable model S. By Theorem 1, the interpretation $S^{\uparrow}$ satisfies the completed definitions of symbols (19). It follows that $S^{\uparrow}$ satisfies the arithmetic completed definitions of these symbols, which are equivalent to axioms (11)–(18). In other words, S describes the extents of the predicates that we want to calculate. Since the only atom in S that begins with b_3 is $b_3(\overline{4},\overline{13})$, the answer to the puzzle is

$$M = 4,\ N = 13.$$

To perform this calculation, we used version 5.6.0 of CLINGO. Earlier versions do not accept the first rule of the program as safe unless the expression:

$$\overline{1} < M \wedge M < N,$$

in the body is rewritten in the interval notation: $M = \overline{2} \mathinner{..} N - \overline{1}$.

5 Proofs

Proofs of Theorems 1 and 2 are based on similar results from an earlier publication (Fandinno *et al.* 2020), and we begin by reviewing them for the special case of regular programs.

5.1 Review: Completion according to Fandinno *et al.*

For any regular term t, the formula $val_t(Z)$, where Z is a general variable that does not occur in t, is defined recursively:

- if t is a numeral or a variable 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 a symbolic constant then $val_t(Z)$ stands for $Z = t$. If $t_1, \dots, t_k$ is a tuple of symbolic constants and regular terms, and $Z_1, \dots, Z_k$ are pairwise distinct general variables that do not occur in $t_1, \dots, t_k$, then $val_{t_1,\dots,t_k}(Z_1, \dots, Z_k)$ stands for $val_{t_1}(Z_1) \wedge \cdots \wedge val_{t_k}(Z_k)$. If t is $t_1 \mathinner{..} t_2$ then $val_t(Z)$ stands for

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

The translation τ^B transforms expressions in the body of a regular rule into formulas 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 $\exists \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \wedge \neg p(\mathbf{Z}))$;
- $\tau^B(t_1 \prec t_2)$ is $\exists Z_1 Z_2(val_{t_1,t_2}(Z_1, Z_2) \wedge Z_1 \prec Z_2)$;
- $\tau^B(t_1 = t_2 \mathinner{..} t_3)$ is $\exists Z_1 Z_2(val_{t_1}(Z_1) \wedge val_{t_2 \mathinner{..} t_3}(Z_2) \wedge Z_1 = Z_2)$;
- $\tau^B(B_1 \wedge B_2 \wedge \cdots)$ is $\tau^B(B_1) \wedge \tau^B(B_2) \wedge \cdots$.

The *completed definition of p/n in Π in the sense of Fandinno et al.* is the sentence over σ_Π constructed as follows. Choose a tuple $\mathbf{V}$ of n general variables that do not occur in Π. For every rule R in the definition D of p/n in Π, by F'_R we denote the formula:

$$\tau^B(Body) \wedge val_{\mathbf{t}}(\mathbf{V})$$

https://doi.org/10.1017/S1471068423000224 Published online by Cambridge University Press

if R is (5), and

$$\tau^B(Body) \wedge val_{\mathbf{t}}(\mathbf{V}) \wedge p(\mathbf{V}),$$

if R is (6). The completed definition of p/n in Π according to Fandinno *et al.* is the sentence:

$$\forall \mathbf{V} \left(p(\mathbf{V}) \leftrightarrow \bigvee_{R \in D} \exists \mathbf{U}'_R F'_R \right),$$

where $\mathbf{U}'_R$ is the list of all variables occurring in R.

For example, the completed definition of one-rule program (3) is

$$\forall V \left(even(V) \leftrightarrow \exists X \left(\exists Z_1 Z_2 \left(Z_1 = X \wedge val_{\overline{-10}\,..\,\overline{10}}(Z_2) \wedge Z_1 = Z_2 \right) \wedge val_{\overline{2} \times X}(V) \right) \right), \tag{21}$$

where $val_{\overline{-10}\,..\,\overline{10}}(Z_2)$ stands for

$$\exists IJK(I = \overline{-10} \wedge J = \overline{10} \wedge I \leq K \leq J \wedge Z_2 = K),$$

and $val_{\overline{2} \times X}(V)$ stands for

$$\exists IJ(V = I \times J \wedge I = \overline{2} \wedge J = X).$$

By COMP(Π), we denote the set of sentences that includes

- for every predicate symbol p/n occurring in Π, its completed definition in Π in the sense of Fandinno *et al.*, and
- for every constraint $\leftarrow Body$ in Π, the universal closure of the formula $\neg\tau^B(Body)$.

Lemma 1
For any regular program Π and any subset S of its Herbrand base, if S is a stable model of Π then $S^{\uparrow}$ satisfies COMP(Π) (Fandinno *et al.* 2020, Theorem 1).

Lemma 2
For any tight regular program Π and any subset S of its Herbrand base, S is a stable model of Π iff $S^{\uparrow}$ satisfies COMP(Π) (Fandinno *et al.* 2020, Theorem 2).

5.2 Main Lemma

Theorems 1 and 2 follow from Lemmas 1 and 2 in view of the following fact proved below:

Main Lemma
For any regular program Π, the formula NCOMP(Π) is equivalent to COMP(Π) in classical predicate calculus with equality.

In the statements of Lemmas 3–7, R is a regular rule, $\mathbf{X}$ is the list of its critical variables, and $\mathbf{I}$ is $f_R(\mathbf{X})$.

Lemma 3
If $\mathbf{t}$ is a list of symbolic constants and regular terms that occur in R, and $\mathbf{Z}$ is a list of pairwise distinct general variables that do not occur in R, then the formula

$$\mathbf{I} = \mathbf{X} \rightarrow \forall \mathbf{Z}(val_{\mathbf{t}}(\mathbf{Z}) \leftrightarrow \mathbf{Z} = f_R(\mathbf{t})), \tag{22}$$

is logically valid (Lifschitz 2021, Lemma 1(i)).

https://doi.org/10.1017/S1471068423000224 Published online by Cambridge University Press

Lemma 4
For any regular atom $p(\mathbf{t})$ occurring in R, the formulas

$$\begin{array}{l}\mathbf{I} = \mathbf{X} \rightarrow (f_R(p(\mathbf{t})) \leftrightarrow \tau^B(p(\mathbf{t}))),\\ \mathbf{I} = \mathbf{X} \rightarrow (f_R(not\ p(\mathbf{t})) \leftrightarrow \tau^B(not\ p(\mathbf{t}))),\end{array}$$

are logically valid (Lifschitz 2021, Lemma 1(iii,iv)).

Lemma 5
For any comparison $t_1 \prec t_2$ occurring in R, the formula

$$\mathbf{I} = \mathbf{X} \rightarrow (f_R(t_1 \prec t_2) \leftrightarrow \tau^B(t_1 \prec t_2)),$$

is logically valid (Lifschitz 2021, Lemma 2).

Lemma 6
For any comparison $t_1 = t_2 \mathinner{..} t_3$ occurring in R, the formula

$$\mathbf{I} = \mathbf{X} \rightarrow (f_R(t_1 = t_2 \mathinner{..} t_3) \leftrightarrow \tau^B(t_1 = t_2 \mathinner{..} t_3)),$$

is logically valid (Lifschitz 2021, Lemma 4).

From Lemmas 4–6, we conclude:

Lemma 7
The formula

$$\mathbf{I} = \mathbf{X} \rightarrow (f_R(Body) \leftrightarrow \tau^B(Body)), \tag{23}$$

where $Body$ is the body of R, is logically valid.

Lemma 8
Let R be a regular rule with the body $B_1 \wedge B_2 \wedge \cdots$.

(a) For every variable X that occurs in B_i in the scope of an arithmetic operation, the formula

$$\tau^B(B_i) \rightarrow \exists I(I = X), \tag{24}$$

is logically valid.

(b) If B_i is a comparison of the form $t_1 = t_2 \mathinner{..} t_3$ then formula (24) is logically valid for every variable X that occurs in B_i.

(Lifschitz 2021, Lemmas 7 and 9).

From Lemma 8, we conclude:

Lemma 9
Let R be a regular rule, and let $\mathbf{X}$ be the list of its critical variables. The formula

$$\tau^B(Body) \rightarrow \exists \mathbf{I}(\mathbf{I} = \mathbf{X}), \tag{25}$$

where $Body$ is the body of R and $\mathbf{I}$ is $f_R(\mathbf{X})$, is logically valid.

In the following lemma, R is a regular rule of form (5) or (6), F_R and $\mathbf{U}_R$ are as defined in Section 3.2, and F'_R and $\mathbf{U}'_R$ are as defined in Section 5.1.

https://doi.org/10.1017/S1471068423000224 Published online by Cambridge University Press

Lemma 10
The formula $\exists \mathbf{U}'_R F'_R$ is equivalent to $\exists \mathbf{U}_R F_R$.

Proof Let $\mathbf{X}$ be the list of all critical variables of R, and let $\mathbf{I}$ be $f_R(\mathbf{X})$. It is sufficient to prove the equivalence:

$$\exists \mathbf{I}\, F'_R \leftrightarrow \exists \mathbf{X}\, F_R.$$

We consider the case when R is (5), so that F_R is

$$\tau^B(Body) \wedge val_{\mathbf{t}}(\mathbf{V}),$$

and F'_R is

$$f_R(Body) \wedge \mathbf{V} = f_R(\mathbf{t}).$$

The case of rule (6) is similar.

Left-to-right: assume $\exists \mathbf{I}(f_R(Body) \wedge \mathbf{V} = f_R(\mathbf{t}))$. Since $\forall \mathbf{I} \exists \mathbf{X}(\mathbf{I} = \mathbf{X})$, we can conclude that

$$\exists \mathbf{I}(\exists \mathbf{X}(\mathbf{I} = \mathbf{X}) \wedge f_R(Body) \wedge \mathbf{V} = f_R(\mathbf{t})).$$

It follows that

$$\exists \mathbf{I}\mathbf{X}(\mathbf{I} = \mathbf{X} \wedge f_R(Body) \wedge \mathbf{V} = f_R(\mathbf{t})),$$

because the critical variables $\mathbf{X}$ do not occur in the formula $f_R(Body) \wedge \mathbf{V} = f_R(\mathbf{t})$. Then $\exists \mathbf{I}\mathbf{X}(\tau^B(Body) \wedge val_{\mathbf{t}}(\mathbf{V}))$ follows using the iniversal closures of (22) and (23). Since the integer variables $\mathbf{I}$ do not occur in $\tau^B(Body) \wedge val_{\mathbf{t}}(\mathbf{V})$, the quantifiers binding $\mathbf{I}$ can be dropped.

Right-to-left: assume $\exists \mathbf{X}(\tau^B(Body) \wedge val_{\mathbf{t}}(\mathbf{V}))$. We can conclude, using the universal closure of (25), that

$$\exists \mathbf{X}(\exists \mathbf{I}(\mathbf{I} = \mathbf{X}) \wedge \tau^B(Body) \wedge val_{\mathbf{t}}(\mathbf{V})).$$

It follows that

$$\exists \mathbf{I}\mathbf{X}(\mathbf{I} = \mathbf{X} \wedge \tau^B(Body) \wedge val_{\mathbf{t}}(\mathbf{V})),$$

because the integer variables $\mathbf{I}$ do not occur in the formula $\tau^B(Body) \wedge val_{\mathbf{t}}(\mathbf{V}))$. Then $\exists \mathbf{I}\mathbf{X}(f_R(Body) \wedge \mathbf{V} = f_R(\mathbf{t}))$ follows using the universal closures of (22) and (23). Since the critical variables $\mathbf{X}$ do not occur in $f_R(Body) \wedge \mathbf{V} = f_R(\mathbf{t})$, the quantifiers binding $\mathbf{X}$ can be dropped.

Lemma 11
If R is a regular constraint $\leftarrow Body$ then the universal closures of the formulas $\neg f_R(Body)$ and $\neg \tau^B(Body)$ are equivalent to each other.

Proof Let $\mathbf{X}$ be the list of all critical variables of R, and let $\mathbf{I}$ be $f_R(\mathbf{X})$. It is sufficient to prove the equivalence:

$$\exists \mathbf{I}\, f_R(Body) \leftrightarrow \exists \mathbf{X}\, \tau^B(Body),$$

because it entails

$$\forall \mathbf{I} \neg f_R(Body) \leftrightarrow \forall \mathbf{X} \neg \tau^B(Body),$$

and consequently entails also the equivalence between the universal closures of $\neg f_R(Body)$ and $\neg \tau^B(Body)$.

https://doi.org/10.1017/S1471068423000224 Published online by Cambridge University Press

Left-to-right: assume $\exists\mathbf{I}\, f_R(Body)$. Since $\forall\mathbf{I}\exists\mathbf{X}(\mathbf{I}=\mathbf{X})$, we can conclude that

$$\exists\mathbf{IX}(\mathbf{I}=\mathbf{X}\wedge f_R(Body)),$$

because the critical variables $\mathbf{X}$ do not occur in $f_R(Body)$. Then $\exists\mathbf{IX}\,\tau^B(Body)$ follows using the iniversal closure of (23). Since the integer variables $\mathbf{I}$ do not occur in $\tau^B(Body)$, the quantifiers binding $\mathbf{I}$ can be dropped.

Right-to-left: assume $\exists\mathbf{X}\,\tau^B(Body)$. We can conclude, using the universal closure of (25), that

$$\exists\mathbf{IX}(\mathbf{I}=\mathbf{X}\wedge \tau^B(Body)),$$

because the integer variables $\mathbf{I}$ do not occur in $\tau^B(Body)$. Then $\exists\mathbf{IX}\, f_R(Body)$ follows using the universal closure of (23). Since the critical variables $\mathbf{X}$ do not occur in $f_R(Body)$, the quantifiers binding $\mathbf{X}$ can be dropped.

Main Lemma follows from Lemmas 10 and 11.

6 Discussion

In the presence of arithmetic operations, the completed definition in the sense of Fandinno *et al.* is often longer and syntactically more complex than the "natural" completed definition introduced in Section 3.2; compare, for instance, formula (21) with (2). On the other hand, the approach of Fandinno *et al.* is applicable to some types of rules that are accepted by CLINGO but are not regular, such as

```
p(1..8,1..8).
p(2*(1..8)).
```

These two rules can be easily rewritten as regular rules:

```
p(X,Y) :- X = 1..8, Y = 1..8.
p(2*X) :- X = 1..8.
```

Regularizing the rule

```
q(X/5) :- p(X).
```

in a similar way gives the rule:

```
q(Y) :- p(X), X = 5*Y+Z, Z = 0..4.
```

that the current version of CLINGO considers unsafe.

The translation COMP is used in the design of the proof assistant ANTHEM (Fandinno *et al.* 2020), and our Main Lemma shows that NCOMP can be employed in the same way. In the process of interacting with ANTHEM, the user often has to read and modify completion formulas. A version of ANTHEM that implements natural completion would make this work easier.

It would be interesting to extend the definition of NCOMP to programs containing conditional literals (Gebser *et al.* 2019, Section 3.1.11). Such an extension would make the reverse completion process applicable to some formulas that are more complex syntactically than the current version. For instance, it may be able to handle the formula:

$$b_1(M,N) \leftrightarrow$$
$$b_0(M,N) \wedge \neg\exists JK(b_0(J,K) \wedge M+N = J+K \wedge \neg puzzling_0(J \times K)),$$

which can replace axioms (13), (14) in the first-order formalization of the Sum and Product Puzzle.

Acknowledgements

Many thanks to Jorge Fandinno, Michael Gelfond, Yuliya Lierler, Jayadev Misra, and Nathan Temple for discussions related to the topic of this paper and to the anonymous referees for advice on improving the previous version.

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 *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Plenum Press, New York, 293–322.

Fages, F. 1991. A fixpoint semantics for general logic programs compared with the well-supported and stable model semantics. *New Generation Computing*, *9*, 425–443.

Fandinno, J., Lifschitz, V., Lühne, P. and Schaub, T. 2020. Verifying tight logic programs with Anthem and Vampire. *Theory and Practice of Logic Programming*, *20*.

Gebser, M., Harrison, A., Kaminski, R., Lifschitz, V. and Schaub, T. 2015. Abstract Gringo. *Theory and Practice of Logic Programming*, *15*, 449–463.

Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T. and Thiele, S. 2019. Potassco User Guide. URL: https://github.com/potassco/guide/releases/.

Lierler, Y. and Maratea, M. 2004. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 346–350.

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

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*.

Lin, F. and Zhao, Y. 2004. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, *157*, 115–137.

Lloyd, J. and Topor, R. 1984. Making Prolog more expressive. *Journal of Logic Programming*, *1*, 225–240.

McCarthy, J. 1990. Formalization of two puzzles involving knowledge. In *Formalizing Common Sense: Papers by John McCarthy*. Ablex, Norwood, NJ.

Misra, J. 2022. *Effective Theories in Programming Practice*. Association for Computing Machinery.

van Ditmarsch, H., Ruan, J. and Verbrugge, L. C. 2005. Model checking sum and product. In *Proceedings of Australasian Joint Conference on Artificial Intelligence*, 790–795.