TPLP: Page 1–31.

doi:10.1017/S147106842300039X

*Locally Tight Programs**

JORGE FANDINNO
University of Nebraska Omaha, USA
(*e-mail:* jorgefandinno@gmail.com)

VLADIMIR LIFSCHITZ and NATHAN TEMPLE
University of Texas at Austin, USA
(*e-mails:* lifschitzv@gmail.com, nathan-temple@live.com)

submitted 30 December 2021; revised 19 October 2023; accepted 15 December 2023

Abstract

Program completion is a translation from the language of logic programs into the language of first-order theories. Its original definition has been extended to programs that include integer arithmetic, accept input, and distinguish between output predicates and auxiliary predicates. For tight programs, that generalization of completion is known to match the stable model semantics, which is the basis of answer set programming. We show that the tightness condition in this theorem can be replaced by a less restrictive "local tightness" requirement. From this fact we conclude that the proof assistant ANTHEM-P2P can be used to verify equivalence between locally tight programs.

KEYWORDS: theory, specification, analysis and verification of systems, security, logic programming methodology and applications

1 Introduction

Program completion (Clark 1978) is a translation from the language of logic programs into the language of first-order theories. If, for example, a program defines the predicate $q/1$ by the rules

$$\begin{aligned} &q(a), \\ &q(X) \leftarrow p(X), \end{aligned} \tag{1}$$

then the completed definition of $q/1$ is

$$\forall X(q(X) \leftrightarrow X = a \vee p(X)). \tag{2}$$

Program (1) and formula (2) express, in different languages, the same idea: the set q consists of a and all elements of p.

Fages (1994) identified a class of programs for which the completion semantics is equivalent to the stable model semantics. Programs in this class are now called *tight*. This

* Many thanks to Yuliya Lierler and to the anonymous referees for their valuable comments.

relationship between completion and stable models plays an important role in the theory of answer set programming (ASP). It is used in the design of the answer set solvers CMODELS (Lierler and Maratea 2004), ASSAT (Lin and Zhao 2004) and CLASP (Gebser *et al.* 2007); and in the design of the proof assistants ANTHEM (Fandinno *et al.* 2020) and ANTHEM-P2P (Fandinno *et al.* 2023).

Some constructs that are common in ASP programs are not covered by Clark's definition of program completion, and that definition had to be generalized. First, the original definition does not cover programs that involve integer arithmetic. To extend it to such programs, we distinguish, in the corresponding first-order formula, between "general" variables on the one hand, and variables for integers on the other (Lifschitz *et al.* 2019, Section 5). This is useful because function symbols in a first-order language are supposed to represent total functions, and arithmetical operations are not defined on symbolic constants.

Second, in ASP programs we often find auxiliary predicate symbols, which are not considered part of its output. Consider, for instance, the program

$$in(P,R,0) \leftarrow in_0(P,R), \tag{3}$$
$$in(P,R,T+1) \leftarrow goto(P,R,T), \tag{4}$$
$$\{\, in(P,R,T+1) \,\} \leftarrow in(P,R,T) \wedge T = 0\,..\,h-1, \tag{5}$$
$$\leftarrow in(P,R_1,T) \wedge in(P,R_2,T) \wedge R_1 \neq R_2, \tag{6}$$
$$in\_building(P,T) \leftarrow in(P,R,T), \tag{7}$$
$$\leftarrow not\ in\_building(P,T) \wedge person(P) \wedge T = 0\,..\,h, \tag{8}$$

which will be used as a running example. It describes the effect of an action – walking from one room to another – on the location of a person. We can read $in(P,R,T)$ as "person P is in room R at time T," and $goto(P,R,T)$ as "person P goes to room R between times T and $T+1$." The placeholder h represents the horizon – the length of scenarios under consideration. Choice rule (5) encodes the commonsense law of inertia for this dynamic system: in the absence of information to the contrary, the location of a person at time $T+1$ is presumed to be the same as at time T. To run this program, we need to specify the value of h and the input predicates $person/1$, $in_0/2$ and $goto/3$. The output is represented by the atoms in the stable model that contain $in/3$. The predicate symbol $in\_building/2$ is auxiliary; a file containing program (3)–(8) may contain a directive that causes the solver not to display the atoms containing that symbol.

For this program and other programs containing auxiliary symbols, we would like the completion to describe the "essential parts" of its stable models, with all auxiliary atoms removed. This can be accomplished by replacing the auxiliary symbols in Clark's completion by existentially quantified predicate variables (Fandinno *et al.* 2020, Section 6.2). The version of completion defined in that paper covers both integer arithmetic and the distinction between the entire stable model and its essential part. Standard models of the completion, in the sense of that paper, correspond to the essential parts of the program's stable models if the program is tight (Fandinno *et al.* 2020, Theorem 2, reproduced in Section 3.3 below).

That theorem is not applicable, however, to program (3)–(8), because this program is not tight. Tightness is defined as the absence of certain cyclical dependencies between

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

predicate symbols (see Section 3.3 for details). The two occurrences of the predicate $in/3$ in rule (5) create a dependency that is not allowed in a tight program.

In this article we propose the definition of a "locally tight" program and show that the above-mentioned theorem by Fandinno et al. can be extended to programs satisfying this less restrictive condition. Local tightness is defined in terms of dependencies between ground atoms. The last argument of $in/3$ in the head of rule (3)–(8) is $T + 1$, and the last argument of $in/3$ in the body is T; for this reason, the dependencies between ground atoms corresponding to this rule are not cyclic. ASP encodings of dynamic systems (Lifschitz 2019, Chapter 8) provide many examples of this kind. The tightness condition prohibits pretty much all uses of recursion in a program; local tightness, on the other hand, expresses the absence of "nonterminating" recursion.

The result of this article shows, for example, that the completion of program (3)–(8) correctly describes the essential parts of its stable models.

Section 2 is a review of definitions and notation related to ASP programs with arithmetic. After defining program completion in Section 3, we introduce locally tight programs and state a theorem describing the relationship between stable models and completion under a local tightness assumption (Section 4). In Section 5, that theorem is used to justify the usability of the proof assistant ANTHEM-P2P for verifying equivalence between programs in some cases that are not allowed in the original publication (Fandinno *et al*. 2023). Proofs of the theorems stated in these sections are based on a lemma of more general nature, which relates stable models to completion for a class of many-sorted first-order theories. This Main Lemma is stated in Section 6 and proved in Section 7. Proofs of the theorems are given in Sections 8–10. In two appendices, we review terminology and notation related to many-sorted formulas and their stable models.

2 Background

This review follows previous publications on ASP programs with arithmetic (Lifschitz *et al*. 2019; Fandinno *et al*. 2020; 2023).

2.1 Programs

The programming language mini-GRINGO, defined in this section, is a subset of the input language of the grounder GRINGO (Gebser *et al*. 2007). Most constructs included in this language are available also in the input language of the solver DLV (Leone *et al*. 2006). The description of mini-GRINGO programs below uses "abstract syntax," which disregards some details related to representing programs by strings of ASCII characters (Gebser *et al*. 2015).

2.1.1 Syntax

We assume that three countably infinite sets of symbols are selected: *numerals*, *symbolic constants*, and *variables*. We assume that a 1-1 correspondence between numerals and integers is chosen; the numeral corresponding to an integer n is denoted by $\overline{n}$. In examples, we take the liberty to drop the overline in numerals. This convention is used, for instance,

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

in rule (5), which should be written, strictly speaking, as

$$\{\, in(P, R, T + \overline{1}) \,\} \leftarrow in(P, R, T) \wedge T = \overline{0} \,..\, h - \overline{1}.$$

Precomputed terms are numerals and symbolic constants. We assume that a total order on precomputed terms is chosen such that for all integers m and n, $\overline{m} < \overline{n}$ iff $m < n$.

Terms allowed in a mini-GRINGO program are formed from precomputed terms and variables using the absolute value symbol $|\,|$ and six binary operation names

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

An *atom* is a symbolic constant optionally followed by a tuple of terms in parentheses. A *literal* is an atom possibly preceded by one or two occurrences of *not*. A *comparison* is an expression of the form t_1 *rel* t_2, where t_1, t_2 are terms and *rel* is $=$ or one of the comparison symbols

$$\neq \quad < \quad > \quad \leq \quad \geq . \tag{9}$$

A *rule* is an expression of the form *Head* $\leftarrow$ *Body*, where

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

A rule is *ground* if it does not contain variables. A ground rule of the form *Head* $\leftarrow$, where *Head* is an atom, will be identified with the atom *Head* and called a *fact*.

A *program* is a finite set of rules.

A *predicate symbol* is a pair p/n, where p is a symbolic constant, and n is a nonnegative integer. We say that p/n *occurs* in a literal l if l contains an atom of the form $p(t_1, \ldots, t_n)$, and similarly for occurrences in rules and in other syntactic expressions.

An atom $p(t_1, \ldots, t_n)$ is *precomputed* if the terms $t_1, \ldots, t_n$ are precomputed.

2.1.2 Semantics

The semantics of ground terms is defined by assigning to every ground term t a finite set $[t]$ of precomputed terms called its *values*. It is recursively defined as follows:

- if t is a numeral or a symbolic constant, then $[t]$ is the singleton set $\{t\}$;
- if t is $|t_1|$, then $[t]$ is the set of numerals $\overline{|n|}$ for all integers n such that $\overline{n} \in [t_1]$;
- if t is $(t_1 + t_2)$, then $[t]$ is the set of numerals $\overline{n_1 + n_2}$ for all integers n_1, n_2 such that $\overline{n_1} \in [t_1]$ and $\overline{n_2} \in [t_2]$; similarly when t is $(t_1 - t_2)$ or $(t_1 \times t_2)$;
- if t is (t_1/t_2), then $[t]$ is the set of numerals $\overline{round(n_1/n_2)}$ for all integers n_1, n_2 such that $n_1 \in [t_1]$, $n_2 \in [t_2]$, and $n_2 \neq 0$;
- if t is $(t_1 \backslash t_2)$, then $[t]$ is the set of numerals $\overline{n_1 - n_2 \cdot round(n_1/n_2)}$ for all integers n_1, n_2 such that $n_1 \in [t_1]$, $n_2 \in [t_2]$, and $n_2 \neq 0$;
- if t is $(t_1..t_2)$, then $[t]$ is the set of numerals $\overline{m}$ for all integers m such that, for some integers n_1, n_2, $n_1 \in [t_1]$, $n_2 \in [t_2]$, $n_1 \leq m \leq n_2$;

where the function *round* is defined as follows:

$$round(n) = \begin{cases} \lfloor n \rfloor & \text{if } n \geq 0, \\ \lceil n \rceil & \text{if } n < 0. \end{cases}$$

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

The use of this function reflects the fact that the grounder GRINGO (Gebser *et al.* 2019) truncates non-integer quotients toward zero. This feature of GRINGO was not taken into account in earlier publications (Gebser *et al.* 2015, Section 4.2; Lifschitz *et al.* 2019, Section 6; Fandinno *et al.* 2020, Section 3), where the floor function was used.

For instance,

$$[\overline{7}/\overline{2}] = \{\overline{3}\},\ [\overline{0}\,..\,\overline{2}] = \{\overline{0},\overline{1},\overline{2}\},\ [\overline{2}/\overline{0}] = [\overline{2}\,..\,\overline{0}] = \emptyset;$$

$[\overline{2}+c] = [\overline{2}\,..\,c] = \emptyset$ if c is a symbolic constant.

For any ground terms $t_1,\dots,t_n$, by $[t_1,\dots,t_n]$ we denote the set of tuples $r_1,\dots,r_n$ for all $r_1 \in [t_1],\dots,r_n \in [t_n]$.

The semantics of programs is defined by rewriting rules in the syntax of propositional logic and referring to the definition of a stable model (answer set) of a propositional theory (Ferraris 2005). The transformation τ is defined as follows. For any ground atom $p(\mathbf{t})$, where $\mathbf{t}$ is a tuple of terms,

- $\tau(p(\mathbf{t}))$ stands for $\bigvee_{\mathbf{r}\in[\mathbf{t}]} p(\mathbf{r})$,
- $\tau(notp(\mathbf{t}))$ stands for $\bigvee_{\mathbf{r}\in[\mathbf{t}]} \neg p(\mathbf{r})$, and
- $\tau(notnotp(\mathbf{t}))$ stands for $\bigvee_{\mathbf{r}\in[\mathbf{t}]} \neg\neg p(\mathbf{r})$.

For any ground comparison $t_1 rel t_2$, we define $\tau(t_1 rel t_2)$ as

- $\top$ if the relation *rel* holds between some r_1 from $[t_1]$ and some r_2 from $[t_2]$;
- $\bot$ otherwise.

If each of $C_1,\dots,C_k$ is a ground literal or a ground comparison, then $\tau(C_1 \wedge \cdots \wedge C_k)$ stands for $\tau C_1 \wedge \cdots \wedge \tau C_k$.

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

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

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

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

If R is a ground constraint $\leftarrow Body$, then τR is $\neg\tau(Body)$.

An *instance* of a rule is a ground rule obtained from it by substituting precomputed terms for variables. For any program Π, $\tau\Pi$ is the set of the formulas τR for all instances R of the rules of Π. Thus $\tau\Pi$ is a set of propositional combinations of precomputed atoms.

For example, τ transforms $\{q(X)\} \leftarrow p(X)$ into the set of formulas

$$p(t) \rightarrow (q(t) \vee \neg q(t)),$$

for all precomputed terms t. The rule $q(\overline{0}\,..\,\overline{2}) \leftarrow not\ p$ is transformed into

$$\neg p \rightarrow (q(\overline{0}) \wedge q(\overline{1}) \wedge q(\overline{2})).$$

Stable models of a program are defined as stable models of the set of formulas obtained from it by applying the transformation τ. Thus stable models of programs are sets of precomputed atoms. This definition is a special case of the definition proposed by Lifschitz *et al.* (2019, Section 3) for Abstract Gringo, except for the changes in the treatment of the division and modulo operations mentioned above.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

2.2 Programs with input and output

A *program with input and output*, or an *io-program*, is a quadruple

$$(\Pi, PH, In, Out), \tag{10}$$

where

- PH is a finite set of symbolic constants;
- In is a finite set of predicate symbols,
- Out is a finite set of predicate symbols that is disjoint from In,

and Π is a mini-GRINGO program such that the symbols from In do not occur in the heads of its rules. Members of PH are the *placeholders* of (10); members of In are the *input symbols* of (10); members of Out are the *output symbols* of (10). The input symbols and output symbols of an io-program are collectively called its *public symbols*. The other predicate symbols occurring in the rules are *private*. An *input atom* is an atom $p(t_1, \dots, t_n)$ such that p/n is an input symbol. *Output atoms*, *public atoms* and *private atoms* are defined in a similar way.

For any set $\mathcal{P}$ of public atoms, $\mathcal{P}^{in}$ stands for the set of input atoms in $\mathcal{P}$.

The example discussed in the introduction corresponds to program (3)–(8) as Π, and

$$PH = \{h\},\ In = \{person/1, in_0/2, goto/3\},\ Out = \{in/3\}. \tag{11}$$

This io-program will be denoted by Ω_1. The only private symbol of Ω_1 is $in\_building/2$.

A *valuation* on a set PH of symbolic constants is a function that maps elements of PH to precomputed terms that do not belong to PH. An *input* for an io-program (10) is a pair $(v, \mathcal{I})$, where

- v is a valuation on the set PH of its placeholdlers, and
- $\mathcal{I}$ is a set of precomputed input atoms such that its members do not contain placeholders.

An input $(v, \mathcal{I})$ represents a way to choose the values of placeholders and the extents of input predicates: for every placeholder c, specify $v(c)$ as its value, and add the atoms $\mathcal{I}$ to the rules of the program as facts. If Π is a mini-GRINGO program then $v(\Pi)$ stands for the program obtained from Π by replacing every occurrence of every constant c in the domain of v by $v(c)$. Using this notation, we can say that choosing $(v, \mathcal{I})$ as input for Π amounts to replacing Π by the program $v(\Pi) \cup \mathcal{I}$.

About a set of precomputed atoms we say that it is an *io-model* of an io-program (10) for an input $(v, \mathcal{I})$ if it is the set of all public atoms of some stable model of the program $v(\Pi) \cup \mathcal{I}$. If $\mathcal{P}$ is an io-model of an io-program for an input $(v, \mathcal{I})$ then $\mathcal{P}^{in} = \mathcal{I}$, because the only rules of $v(\Pi) \cup \mathcal{I}$ containing input symbols in the head are the facts $\mathcal{I}$.

For example, an input $(v, \mathcal{I})$ for the io-program Ω_1 can be defined by the conditions

$$\begin{aligned} &v(h) = \overline{2}, \\ &\mathcal{I} = \{person(alice), person(bob), \\ &\qquad in_0(alice, hall), in_0(bob, hall), \\ &\qquad goto(alice, classroom, \overline{0}), goto(bob, classroom, \overline{1})\}. \end{aligned} \tag{12}$$

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

The program $v(\Pi) \cup \mathcal{I}$ has a unique stable model, which consists of the members of $\mathcal{I}$, the atoms

$$\begin{array}{l}\{in(alice, hall, \overline{0}), in(bob, hall, \overline{0}),\\ \ in(alice, classroom, \overline{1}), in(bob, hall, \overline{1}),\\ \ in(alice, classroom, \overline{2}), in(bob, classroom, \overline{2})\},\end{array} \tag{13}$$

and the private atoms $in\_building(p, \overline{i})$ for all p in $\{alice, bob\}$ and all i in $\{0, 1, 2\}$. The io-model of Ω_1 for $(v, \mathcal{I})$ consists of the members of $\mathcal{I}$ and atoms (13).

2.3 Two-sorted formulas and standard interpretations

The two-sorted signature σ_0 includes

- the sort *general* and its subsort *integer*;
- all precomputed terms of mini-GRINGO as object constants; an object constant is assigned the sort *integer* iff it is a numeral;
- the symbol $||$ as a unary function constant; its argument and value have the sort *integer*;
- the symbols $+$, $-$ and $\times$ as binary function constants; their arguments and values have the sort *integer*;
- predicate symbols p/n as n-ary predicate constants; their arguments have the sort *general*;
- symbols (9) as binary predicate constants; their arguments have the sort *general*.[1]

A formula of the form $(p/n)(\mathbf{t})$ can be written also as $p(\mathbf{t})$. This convention allows us to view precomputed atoms (Section 2.1.1) as sentences over σ_0. Conjunctions of equalities and inequalities can be abbreviated as usual in algebra; for instance, $X = Y < Z$ stands for $X = Y \wedge Y < Z$.

An interpretation of the signature σ_0 is *standard* if

(a) its domain of the sort *general* is the set of precomputed terms;
(b) its domain of the sort *integer* is the set of numerals;
(c) every object constant represents itself;
(d) the absolute value symbol and the binary function constants are interpreted as usual in arithmetic;
(e) predicate constants (9) are interpreted in accordance with the total order on precomputed terms chosen in the definition of mini-GRINGO (Section 2.1.1).

A standard interpretation will be uniquely determined if we specify the set $\mathcal{J}$ of precomputed atoms to which it assigns the value *true*. This interpretation will be denoted by $\mathcal{J}^{\uparrow}$.

When formulas over σ_0 are used in reasoning about io-programs with placeholders, we may need interpretations satisfying a condition different from (c). An interpretation is *standard for* a set PH of symbolic constants if it satisfies conditions (a), (b), (d), (e) above and the conditions

[1] The symbols / and \ are not included in this signature because division is not a total function on integers. The symbol .. is not included either, because intervals are not meant to be among values of variables of this first-order language.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

(c′) every object constant that belongs to PH represents a precomputed term that does not belong to PH;
(c″) every object constant that does not belong to PH represents itself.

An interpretation that is standard for PH will be uniquely determined if we specify (i) the valuation v that maps every element of PH to the precomputed term that it represents, and (ii) the set $\mathcal{J}$ of precomputed atoms to which it assigns the value *true*. This interpretation will be denoted by $\mathcal{J}^v$.

2.4 Representing rules by formulas

From now on, we assume that every symbol designated as a variable in mini-GRINGO is among general variables of the signature σ_0. The transformation τ^*, described in this section, converts mini-GRINGO rules into formulas over σ_0.

First we define, for every mini-GRINGO term t, a formula $val_t(V)$ over the signature σ_0, where V is a general variable that does not occur in t. That formula expresses, informally speaking, that V is one of the values of t. The definition is recursive:[2]

- if t is a precomputed term or a variable then $val_t(V)$ is $V = t$,
- if t is $|t_1|$ then $val_t(V)$ is $\exists I(val_{t_1}(I) \wedge V = |I|)$,
- if t is $t_1 \mathbin{op} t_2$, where op is $+$, $-$, or $\times$ then $val_t(V)$ is

$$\exists IJ(val_{t_1}(I) \wedge val_{t_2}(J) \wedge V = I \mathbin{op} J),$$

- if t is $t_1 \mathbin{/} t_2$ then $val_t(V)$ is

$$\begin{aligned}\exists IJK(&val_{t_1}(I) \wedge val_{t_2}(J) \wedge K \times |J| \leq |I| < (K + \overline{1}) \times |J| \\ &\wedge ((I \times J \geq \overline{0} \wedge V = K) \vee (I \times J < \overline{0} \wedge V = -K))),\end{aligned}$$

- if t is $t_1 \backslash t_2$ then $val_t(V)$ is

$$\begin{aligned}\exists IJK(&val_{t_1}(I) \wedge val_{t_2}(J) \wedge K \times |J| \leq |I| < (K + \overline{1}) \times |J| \\ &\wedge ((I \times J \geq \overline{0} \wedge V = I - K \times J) \vee (I \times J < \overline{0} \wedge V = I + K \times J))),\end{aligned}$$

- if t is $t_1 \mathbin{..} t_2$ then $val_t(V)$ is

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

  where I, J, K are fresh integer variables.

If $\mathbf{t}$ is a tuple $t_1, \dots, t_n$ of mini-GRINGO terms, and $\mathbf{V}$ is a tuple $V_1, \dots, V_n$ of distinct general variables, then $val_{\mathbf{t}}(\mathbf{V})$ stands for the conjunction $val_{t_1}(V_1) \wedge \cdots \wedge val_{t_n}(V_n)$.

The next step is to define the transformation τ^B, which converts literals and comparisons into formulas over σ_0. (The superscript B reflects the fact that this translation is close to the meaning of expressions in *bodies* of rules.) It transforms

- $p(\mathbf{t})$ into $\exists \mathbf{V}(val_{\mathbf{t}}(\mathbf{V}) \wedge p(\mathbf{V}))$;
- *not* $p(\mathbf{t})$ into $\exists \mathbf{V}(val_{\mathbf{t}}(\mathbf{V}) \wedge \neg p(\mathbf{V}))$;
- *not not* $p(\mathbf{t})$ into $\exists \mathbf{V}(val_{\mathbf{t}}(\mathbf{V}) \wedge \neg\neg p(\mathbf{V}))$;
- $t_1 \mathbin{rel} t_2$ into $\exists V_1 V_2(val_{t_1}(V_1) \wedge val_{t_2}(V_2) \wedge V_1 \mathbin{rel} V_2)$.

[2] The treatment of division here follows the paper (Fandinno and Lifschitz 2023) that corrects a mistake found in previous publications; see Section 2.1.2.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

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

Now we are ready to define the transformation τ^*. It converts a basic rule

$$p(\mathbf{t}) \leftarrow Body, \tag{14}$$

of a program Π into the formula

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

where $\mathbf{V}$ is the list of alphabetically first general variables that do not occur in Π, and $\widetilde{\forall}$ denotes universal closure (see Appendix A). A choice rule

$$\{p(\mathbf{t})\} \leftarrow Body, \tag{15}$$

is converted into[3]

$$\widetilde{\forall}(val_{\mathbf{t}}(\mathbf{V}) \wedge \tau^B(Body) \wedge \neg\neg p(\mathbf{V}) \rightarrow p(\mathbf{V})), \tag{16}$$

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

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

The transformation τ^* can be used in place of the transformation τ (Section 2.1.2) to describe the stable models of a mini-GRINGO program, as shown by the theorem below. Its statement refers to stable models of many-sorted theories (Appendix B) for the case when the underlying signature is σ_0, and all comparison symbols are classified as extensional. Its proof is given in Section 8.

Theorem 1
A set $\mathcal{J}$ of precomputed atoms is a stable model of a mini-GRINGO program Π iff $\mathcal{J}^\uparrow$ is a stable model of $\tau^*\Pi$.

3 Completion

This section describes the process of constructing the completion of an io-program. This construction involves the formulas $\tau^* R$ for the rules R of the program, and it exploits the special syntactic form of these formulas, as discussed below.

3.1 Completable sets

Consider a many-sorted signature σ (see Appendix A) with its predicate constants partitioned into *intensional* symbols and *extensional* symbols, so that the set of intensional symbols is finite. A sentence over σ is *completable* if it has the form

$$\widetilde{\forall}(F \rightarrow G), \tag{17}$$

[3] In some publications, the result of applying τ^* to a choice rule (15) is defined as

$$\widetilde{\forall}(val_{\mathbf{t}}(\mathbf{V}) \wedge \tau^B(Body) \rightarrow \neg p(\mathbf{V}) \vee \neg p(\mathbf{V})).$$

The difference between this formula and formula (16) is not essential, because the two formulas are satisfied by the same HT-interpretations (see Appendix B).

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

where G either contains no intensional symbols or has the form $p(\mathbf{V})$, where p is an intensional symbol with argument sorts $s_1, \dots, s_n$, and $\mathbf{V}$ is a tuple of pairwise distinct variables of these sorts. A finite set Γ of sentences is *completable* if

- every sentence in Γ is completable, and
- for any two sentences $\widetilde{\forall}(F_1 \to G_1)$, $\widetilde{\forall}(F_2 \to G_2)$ in Γ such that G_1 and G_2 contain the same intensional symbol, $G_1 = G_2$.

It is easy to see that for any mini-GRINGO program Π, the set $\tau^*\Pi$ is completable.

The *definition* of an intensional symbol p in a completable set Γ is the set of members (17) of Γ such that p occurs in G. A *first-order constraint* is a completable sentence (17) such that G does not contain intensional symbols. Thus Γ is the union of the definitions of intensional symbols and a set of first-order constraints.

If the definition of p in Γ consists of the sentences $\widetilde{\forall}(F_i(\mathbf{V}) \to p(\mathbf{V}))$ then the *completed definition* of p in Γ is the sentence

$$\forall \mathbf{V} \left(p(\mathbf{V}) \leftrightarrow \bigvee_i \exists \mathbf{U}_i \, F_i(\mathbf{V}) \right),$$

where $\mathbf{U}_i$ is the list of free variables of $F_i(\mathbf{V})$ that do not belong to $\mathbf{V}$.

The *completion* COMP[Γ] of a completable set Γ is the conjunction of the completed definitions of all intensional symbols of σ in Γ and the constraints of Γ.

3.2 Completion of a program with input and output

The *completion* of an io-program (10) is the second-order sentence[4] $\exists P_1 \cdots P_l \, C$, where C is obtained from the sentence COMP[$\tau^*\Pi$] by replacing all private symbols $p_1/n_1, \dots, p_l/n_l$ with distinct predicate variables $P_1, \dots, P_l$. The completion of Ω will be denoted by COMP[Ω].[5] In building COMP[Ω], we classify as extensional all comparison and input symbols. All output and private symbols are intensional.

Consider, for example, the io-program with the rules

$$\begin{aligned} &p(a), \\ &p(b), \\ &q(X,Y) \leftarrow p(X) \wedge p(Y), \end{aligned} \tag{18}$$

without input symbols and with the output symbol $q/2$. The translation τ^* transforms these rules into the formulas

$$\begin{aligned} &\forall V_1 (V_1 = a \to p(V_1)), \\ &\forall V_1 (V_1 = b \to p(V_1)), \\ &\forall XY V_1 V_2 (V_1 = X \wedge V_2 = Y \wedge \exists V (V = X \wedge p(V)) \wedge \exists V (V = Y \wedge p(V)) \to q(V_1, V_2)). \end{aligned} \tag{19}$$

[4] Recall that second-order formulas may contain predicate variables, which can be used to form atomic formulas in the same way as predicate constants, and can be bound by quantifiers in the same way as object variables (Lifschitz *et al.* 2008, Section 1.2.3). In many-sorted setting, each predicate variable is assigned argument sorts, like a predicate constant (see Appendix A).

[5] The definitions of COMP[Ω] in other publications do not refer to the translation τ^*. But they describe formulas that are almost identical to the sentence COMP[Ω] defined here, and are equivalent to it in classical second-order logic.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

The completed definition of $p/1$ in (19) is

$$\forall V_1(p(V_1) \leftrightarrow V_1 = a \vee V_1 = b),$$

and the completed definition of $q/2$ is

$$\forall V_1 V_2(q(V_1, V_2) \leftrightarrow \exists XY(V_1 = X \wedge V_2 = Y \wedge \exists V(V = X \wedge p(V)) \wedge \exists V(V = Y \wedge p(V)))).$$

The completion of the program is

$$\begin{aligned}&\exists P(\forall V_1(P(V_1) \leftrightarrow V_1 = a \vee V_1 = b) \wedge \\ &\quad \forall V_1 V_2(q(V_1, V_2) \leftrightarrow \\ &\qquad \exists XY(V_1 = X \wedge V_2 = Y \wedge \exists V(V = X \wedge P(V)) \wedge \exists V(V = Y \wedge P(V))))).\end{aligned} \tag{20}$$

Formula (20) can be equivalently rewritten as

$$\exists P(\forall V_1(P(V_1) \leftrightarrow V_1 = a \vee V_1 = b) \wedge \forall V_1 V_2(q(V_1, V_2) \leftrightarrow P(V_1) \wedge P(V_2))).$$

Second-order quantifiers in completion formulas can often be eliminated. For instance, formula (20) is equivalent to the first-order formula

$$\forall V_1 V_2(q(V_1, V_2) \leftrightarrow (V_1 = a \vee V_1 = b) \wedge (V_2 = a \vee V_2 = b)).$$

3.3 Review: Tight programs

We are interested in cases when the sentence COMP[Ω] can be viewed as a description of the io-models of Ω. This is a review of one result of this kind.

The *positive predicate dependency graph* of a mini-GRINGO program Π is the directed graph defined as follows. Its vertices are the predicate symbols p/n that occur in Π. It has an edge from p/n to p'/n' iff p/n occurs in the head of a rule R of Π such that p'/n' occurs in an atom that is a conjunctive term of the body of R.

A mini-GRINGO program is *tight* if its positive predicate dependency graph is acyclic. For example, the positive predicate dependency graph of program (18) has a single edge, from $q/2$ to $p/1$; this program is tight. The positive predicate dependency graph of program (3)–(8) includes a self-loop at $in/3$; this program is not tight.

Let Ω be an io-program (Π, PH, In, Out), and let $\mathcal{P}$ be a set of precomputed public atoms that do not contain placeholders of Ω. Under the assumption that Π is tight, $\mathcal{P}$ is an io-model of Ω for an input $(v, \mathcal{I})$ iff $\mathcal{P}^v$ satisfies COMP[Ω] and $\mathcal{P}^{in} = \mathcal{I}$ (Fandinno *et al*. 2020, Theorem 2).

4 Locally tight programs

For any tuple $t_1, \dots, t_n$ of ground terms, $[t_1, \dots, t_n]$ stands for the set of tuples $r_1, \dots, r_n$ of precomputed terms such that $r_1 \in [t_1], \dots, r_n \in [t_n]$.

The *positive dependency graph of an io-program* Ω *for an input* $(v, \mathcal{I})$ is the directed graph defined as follows. Its vertices are the ground atoms $p(r_1, \dots, r_n)$ such that p/n is an output symbol or a private symbol of Ω, and each r_i is a precomputed term different from the placeholders of Ω. It has an edge from $p(\mathbf{r})$ to $p'(\mathbf{r}')$ iff there exists a ground instance R of one of the rules of $v(\Pi)$ such that

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

(a) the head of R has the form $p(\mathbf{t})$ or $\{p(\mathbf{t})\}$, where $\mathbf{t}$ is a tuple of terms such that $\mathbf{r} \in [\mathbf{t}]$;
(b) the body of R has a conjunctive term of the form $p'(\mathbf{t})$, where $\mathbf{t}$ is a tuple of terms such that $\mathbf{r}' \in [\mathbf{t}]$;
(c) for every conjunctive term of the body of R that contains an input symbol of Ω and has the form $q(\mathbf{t})$ or *not not* $q(\mathbf{t})$, the set $[\mathbf{t}]$ contains a tuple $\mathbf{r}$ such that $q(\mathbf{r}) \in \mathcal{I}$;
(d) for every conjunctive term of the body of R that contains an input symbol of Ω and has the form *not* $q(\mathbf{t})$, the set $[\mathbf{t}]$ contains a tuple $\mathbf{r}$ such that $q(\mathbf{r}) \notin \mathcal{I}$;
(e) for every comparison $t_1 \prec t_2$ in the body of R, there exist terms r_1, r_2 such that $r_1 \in [t_1]$, $r_2 \in [t_2]$, and the relation $\prec$ holds for the pair (r_1, r_2).

Recall that an *infinite walk* in a directed graph is an infinite sequence of edges which joins a sequence of vertices. If the positive dependency graph of Ω for an input $(v, \mathcal{I})$ has no infinite walks, then we say that Ω is *locally tight* on that input.

Consider, for example, the positive dependency graph of the io-program Ω_1 (Section 2.2). Its vertices for an input $(v, \mathcal{I})$ are atoms $in(p, r, t)$ and $in\_building(p, t)$, where p, r, t are precomputed terms different from h. The only rules of Ω_1 that contribute edges to the graph are (5) and (7), because all predicate symbols in the bodies of (3) and (4) are input symbols, and rules (6) and (8) are constraints. If R is (5) then $v(R)$ is the rule

$$\{in(P, R, T + \overline{1})\} \leftarrow in(P, R, T) \wedge T = \overline{0} \mathinner{..} v(h) - \overline{1}.$$

An instance

$$\{in(p, r, t + \overline{1})\} \leftarrow in(p, r, t) \wedge t = \overline{0} \mathinner{..} v(h) - \overline{1},$$

of this rule contributes an edge if and only if $v(h)$ is a numeral $\overline{n}$, t is a numeral $\overline{i}$, and $0 \leq i < n$. If R is (7) then $v(R) = R$, and an instance

$$in\_building(p, t) \leftarrow in(p, r, t),$$

of $v(R)$ contributes the edge

$$in\_building(p, t) \longrightarrow in(p, r, t). \tag{21}$$

Consequently the edges of the positive dependency graph of Ω_1 are (21) and

$$in(p, r, \overline{i + 1}) \longrightarrow in(p, r, \overline{i}),$$

if $v(h) = \overline{n}$ and $0 \leq i < n$. It is clear that this graph has no infinite walks. Consequently Ω_1 is locally tight on all inputs.

Proposition 1
If a mini-GRINGO program Π is tight then any io-program of the form (Π, PH, In, Out) is locally tight for all inputs.

Proof
Assume that Π is tight, but the positive dependency graph of (Π, PH, In, Out) for some input has an infinite walk. If that graph has an edge from $p(r_1, \dots, r_n)$ to $p'(r'_1, \dots, r'_{n'})$ then the positive predicate dependency graph of Π has an edge from p/n to p'/n'. Consequently this graph has an infinite walk as well. But that is impossible, because this graph is finite and acyclic. □

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

The example of program Ω_1 shows that the converse of Proposition 1 does not hold.

The theorem below generalizes the property of tight programs reviewed at the end of Section 3.3. For any formula F over σ_0 and any valuation v, $v(F)$ stands for the formula obtained from F by replacing every occurrence of every constant c in the domain of v by $v(c)$. The proof of the following theorem is given in Section 9.

Theorem 2
For any io-program Ω, any set $\mathcal{P}$ of precomputed public atoms that do not contain placeholders of Ω, and any input $(v, \mathcal{I})$ for which Ω is locally tight, the following conditions are equivalent:

(a) $\mathcal{P}$ is an io-model of Ω for $(v, \mathcal{I})$,
(b) $\mathcal{P}^\uparrow$ satisfies $v(\mathrm{COMP}[\Omega])$ and $\mathcal{P}^{in} = \mathcal{I}$,
(c) $\mathcal{P}^v$ satisfies $\mathrm{COMP}[\Omega]$ and $\mathcal{P}^{in} = \mathcal{I}$.

Take, for instance, the io-program Ω_1 and the input $(v, \mathcal{I})$ defined by conditions (12). We observed in Section 2.2 that the set $\mathcal{P}$ obtained from $\mathcal{I}$ by adding atoms (13) is an io-model of Ω_1 for $(v, \mathcal{I})$. According to Theorem 2, this observation is equivalent to the claim that the corresponding interpretation $\mathcal{P}^\uparrow$ satisfies the sentence obtained from $\mathrm{COMP}[\Omega_1]$ by substituting $\overline{2}$ for the placeholder h.

5 Equivalence of io-programs

We begin with an example. Consider the io-program Ω_2, obtained from Ω_1 by replacing inertia rule (5) with the pair of rules

$$go(P, T) \leftarrow goto(P, R, T), \tag{22}$$

$$in(P, R, T + 1) \leftarrow in(P, R, T) \wedge not\ go(P, T) \wedge T = 0\,..\,h - 1. \tag{23}$$

Here $go/2$ is a new private symbol. These rules express commonsense inertia in a different way, using the fact that *goto* actions are the only possible causes of change in the location of a person. Programs Ω_1 and Ω_2 describe the same dynamic system using two different approaches to encoding inertia.

Are these programs equivalent, in some sense? The idea of equivalence with respect to a user guide (Fandinno *et al.* 2023) allows us to make this question precise. We present the details below.

About two io-programs we say that they are *comparable* to each other if they have the same placeholders, the same input symbols, and the same output symbols. For example, Ω_1 and Ω_2 are comparable. Any two comparable programs have the same inputs, because the definition of an input for Ω (Section 2.2) refers to only two components of Ω – to its placeholders and its input symbols.

We will define when comparable io-programs Ω, Ω' are equivalent to each other on a subset *Dom* of their inputs. Sets of inputs will be called *domains*. We need domains to express the idea that two io-programs can be equivalent even if they exhibit different behaviors on some inputs that we consider irrelevant. In an input for Ω_1 or Ω_2, for example, the first argument of each of the predicate symbols $in_0/2$, $goto/3$ is expected to be a person; we are not interested in inputs that contain $in_0(alice, hall)$ but do not contain $person(alice)$.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

Domains are infinite sets, but there is a natural way to represent some domains by finite expressions. An *assumption* for an io-program Ω is a sentence over σ_0 such that every predicate symbol p/n occurring in it is an input symbol of Ω. The domain *defined* by an assumption A is the set of all inputs $(v, \mathcal{I})$ such that the interpretation $\mathcal{I}^v$ satisfies A. For example, the assumption

$$\forall PR(in_0(P, R) \rightarrow person(P)), \tag{24}$$

where P and R are general variables, defines the set $\mathcal{I}$ of all inputs for Ω_1 such that $person(p) \in \mathcal{I}$ whenever $in_0(p, r) \in \mathcal{I}$.

About comparable io-programs Ω, Ω' we say that they are *equivalent* on a domain *Dom* if, for every input $(v, \mathcal{I})$ from *Dom*, Ω and Ω' have the same io-models for that input.

ANTHEM-P2P (Fandinno *et al.* 2023) is a proof assistant designed for verifying equivalence of io-programs in the sense of this definition. It uses the following terminology. A *user guide* is a quadruple

$$(PH, In, Out, Dom), \tag{25}$$

where *PH*, *In* and *Out* are as in the definition of an io-program (Section 2.2), and *Dom* is a domain. The assertion that io-programs (Π, PH, In, Out) and (Π', PH, In, Out) are equivalent on a domain *Dom* can be expressed by saying that Π and Π' are equivalent with respect to user guide (25).[6]

We can say, for example, that replacing rule (5) in mini-GRINGO program (3)–(8) with rules (22) is an equivalent transformation for the user guide (25) with *PH*, *In* and *Out* defined by formulas (11), and with *Dom* defined by the conjunction of assumption (24) and the assumption

$$\forall PRT(goto(P, R, T) \rightarrow person(P)). \tag{26}$$

Theorem 3 below shows that using the ANTHEM-P2P algorithm for verifying equivalence of io-programs can be justified under a local tightness condition. If an io-program is locally tight for all inputs satisfying an assumption A then we say that it is locally tight *under the assumption* A. If two comparable io-programs are equivalent on the domain defined by an assumption A then we say that they are equivalent *under the assumption* A. The proof of the following theorem is given in Section 10.

Theorem 3
Let Ω, Ω' be comparable io-programs with placeholders *PH* that are locally tight under an assumption A. Programs Ω, Ω' are equivalent to each other under the assumption A iff the sentence

$$A \rightarrow (\text{COMP}[\Omega] \leftrightarrow \text{COMP}[\Omega']), \tag{27}$$

is satisfied by all interpretations that are standard for *PH*.

[6] The definition in the ANTHEM-P2P paper looks different from the one given here, but the two definitions are equivalent. Instead of the condition "Ω and Ω' have the same io-models" Fandinno et al. require that Π and Π' have the same external behavior. The external behavior of a mini-GRINGO program Π for a user guide (25) and an input $(v, \mathcal{I})$ can be described as the collection of all sets that can be obtained from io-models of (Π, PH, In, Out) by removing the input atoms $\mathcal{I}$.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

From Theorem 3 we can conclude that equivalence between Ω and Ω' will be established if formula (27) is derived from a set of axioms that are satisfied by all interpretations that are standard for *PH*. This is how ANTHEM-P2P operates (Fandinno *et al.* 2023, Sections 7).

The claim that Ω_1 is equivalent to Ω_2 assuming (24) and (26) has been verified by ANTHEM-P2P, with the resolution theorem prover VAMPIRE (Kovaćs and Voronkov 2013) used as the proof engine. We saw in Section 4 that Ω_1 is locally tight for all inputs. The positive dependency graph of Ω_2 has additional edges, from $go(p,t)$ to $goto(p,r,t)$. It is clear that this graph has no infinite walks either, so that Ω_2 is locally tight for all inputs as well.

The process of verifying equivalence in this case was interactive – we helped VAMPIRE find a proof by suggesting useful lemmas and an induction axiom, as usual in such experiments (Fandinno *et al.* 2023, Section 6; Fandinno *et al.* 2020, Figure 5). Induction was needed to prove the lemma

$$\forall PRT(in(P,R,T) \to person(P)),$$

with (24) used to justify the base case.

6 Main Lemma

The Main Lemma, stated below, relates the completion of a completable set of sentences (Section 3.1) to its stable models (Appendix B).

As in Section 3.1, σ stands here for a many-sorted signature with its predicate constants partitioned into intensional and extensional. The symbols σ^I, $\mathbf{d}^*$, used in this section, are introduced in Appendix A; $I^\downarrow$ is defined in Appendix B.

We define, for an interpretation I of σ and a sentence F over σ^I, the set $Pos(F,I)$ of *(strictly) positive atoms of F with respect to I*. Elements of this set are formulas of the form $p(\mathbf{d}^*)$. This set is defined recursively, as follows. If F does not contain intensional symbols or is not satisfied by I then $Pos(F,I) = \emptyset$. Otherwise,

(i) $Pos(p(\mathbf{t}),I) = \{p((\mathbf{t}^I)^*)\}$;
(ii) $Pos(F_1 \wedge F_2, I) = Pos(F_1 \vee F_2, I) = Pos(F_1,I) \cup Pos(F_2,I)$;
(iii) $Pos(F_1 \to F_2, I) = Pos(F_2, I)$;
(iv) $Pos(\forall X F(X), I) = Pos(\exists X F(X), I) = \bigcup_{d \in |I|^s} Pos(F(d^*), I)$ if X is a variable of sort s.

It is easy to check by induction on F that $Pos(F,I)$ is a subset of the set $I^\downarrow$.

An *instance* of a completable sentence $\widetilde{\forall}(F \to G)$ over σ^I is a sentence obtained from $F \to G$ by substituting names d^* for its free variables. For any completable set Γ of sentences over σ^I, the *positive dependency graph* $G^{sp}_I(\Gamma)$ is the directed graph defined as follows. Its vertices are elements of $I^\downarrow$. It has an edge from A to B iff, for some instance $F \to G$ of a member of Γ, $A \in Pos(G,I)$ and $B \in Pos(F,I)$.

Main Lemma
For any interpretation I of σ and any completable set Γ of sentences over σ^I such that the graph $G^{sp}_I(\Gamma)$ has no infinite walks, I is a stable model of Γ iff $I \models \mathrm{COMP}[\Gamma]$.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

Consider, for example, the signature consisting of a single sort, the object constants a, b, and the intensional unary predicate constant p. Let Γ be

$$\{\forall V(V = a \to p(V)),\ \forall V(V = b \wedge p(V) \to p(V))\},$$

and let the interpretations I, J be defined by the conditions

$$|I| = |J| = \{0, 1, \dots\};\ a^I = a^J = 0;\ b^I = b^J = 1;$$
$$p^I(n) = \mathit{true} \text{ iff } n = 0;\ p^J(n) = \mathit{true} \text{ iff } n \in \{0, 1\}.$$

Instances of the members of Γ are implications of the forms

$$n^* = a \to p(n^*) \quad \text{and} \quad n^* = b \wedge p(n^*) \to p(n^*),$$

$(n = 0, 1, \dots)$. The set $I^{\downarrow}$ of vertices of the graph $G^{sp}_I(\Gamma)$ is $\{p(0^*)\}$. This graph has no edges, because, for every n,

$$Pos(n^* = a, I) = Pos(n^* = b \wedge p(n^*), I) = \emptyset.$$

The set $J^{\downarrow}$ of vertices of $G^{sp}_J(\Gamma)$ is $\{p(0^*), p(1^*)\}$. This graph has one edge – the self-loop at $p(1^*)$, because

$$Pos(1^* = b \wedge p(1^*), J) = Pos(p(1^*), J) = \{p(1^*)\}.$$

Consequently the Main Lemma is applicable to I, but not to J. It asserts that I is a stable model of Γ (which is true) iff I satisfies the completion

$$\forall V(p(V) \leftrightarrow V = a \vee (V = b \wedge p(b))),$$

of Γ (which is true as well). The interpretation J, on the other hand, satisfies the completion of Γ, but it is not a stable model.

The Main Lemma is similar to two results published earlier (Ferraris *et al.* 2011, Theorem 11; Lee and Meng 2011, Corollary 4). The former refers to dependencies between predicate constants, rather than ground atoms. The latter is applicable to formulas of different syntactic form.

About an interpretation I of σ we say that it is *semi-Herbrand* if

- all elements of its domains are object constants of σ, and
- $d^I = d$ for every such element d.[7]

For example, every interpretation of σ_0 that is standard, or standard for some set PH (Section 2.3), is semi-Herbrand.

If I is semi-Herbrand then the graph $G^{sp}_I(\Gamma)$ can be modified by replacing each vertex $p(\mathbf{d}^*)$ by the atom $p(\mathbf{d})$ over the signature σ. This modified positive dependency graph is isomorphic to $G^{sp}_I(\Gamma)$ as defined above. It can be defined independently, by replacing clauses (i) and (iv) in the definition of $Pos(, I)F$ above by

(i)′ $Pos(p(\mathbf{t}), I) = \{p(\mathbf{t}^I)\}$;
(iv)′ $Pos(\forall X F(X), I) = Pos(\exists X F(X), I) = \bigcup_{d \in |I|^s} Pos(F(d), I)$ if X is a variable of sort s.

[7] This is weaker than the condition defining Herbrand interpretations; some ground terms over σ may be different from elements of the domains $|I|^s$.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

7 Proof of the Main Lemma

The Main Lemma expresses a property of the completion operator, and its proof below consists of two parts. We first prove a similar property of pointwise stable models, defined in Appendix B (Lemma 3); then we relate pointwise stable models to completion.

As in Section 6, σ stands for a many-sorted signature with its predicate constants subdivided into intensional and extensional.

Lemma 1
For any HT-interpretation $\langle \mathcal{H}, I\rangle$ and any sentence F over σ^I, if $I \models F$ and $Pos(F, I) \subseteq \mathcal{H}$ then $\langle \mathcal{H}, I\rangle \models_{ht} F$.

Proof
By induction on the size of F. *Case 1*: F does not contain intensional symbols. The assertion of the lemma follows from Proposition 3(b) in Appendix B. *Case 2*: F contains an intensional symbol. Since $I \models F$, the set $Pos(F, I)$ is determined by the recursive clauses in the definition of *Pos*. *Case 2.1:* F is $p(\mathbf{t})$, where p is intensional. Then, the assumption $Pos(F, I) \subseteq \mathcal{H}$ and the claim $\langle \mathcal{H}, I\rangle \models_{ht} F$ turn into the condition $p((\mathbf{t}^I)^*) \in \mathcal{H}$. *Case 2.2:* F is $F_1 \wedge F_2$. Then, from the assumption $I \models F$ we conclude that $I \models F_i$ for $i = 1, 2$. On the other hand,

$$Pos(F_i, I) \subseteq Pos(F, I) \subseteq \mathcal{H}.$$

By the induction hypothesis, it follows that $\langle \mathcal{H}, I\rangle \models_{ht} F_i$, and consequently $\langle \mathcal{H}, I\rangle \models_{ht} F$. *Case 2.3:* F is $F_1 \vee F_2$. Similar to Case 2.2. *Case 2.4:* F is $F_1 \to F_2$. Since $I \models F_1 \to F_2$, we only need to check that $\langle \mathcal{H}, I\rangle \not\models_{ht} F_1$ or $\langle \mathcal{H}, I\rangle \models_{ht} F_2$. *Case 2.4.1:* $I \models F_1$. Since $I \models F_1 \to F_2$, it follows that $I \models F_2$. On the other hand,

$$Pos(F_2, I) = Pos(F, I) \subseteq \mathcal{H}.$$

By the induction hypothesis, it follows that $\langle \mathcal{H}, I\rangle \models_{ht} F_2$. *Case 2.4.2:* $I \not\models F_1$. By Proposition 3(a) in Appendix B, it follows that $\langle \mathcal{H}, I\rangle \not\models_{ht} F_1$. *Case 2.5:* F is $\forall X\, G(X)$, where X is a variable of sort s. Then, for every element d of $|I|^s$, $I \models G(d^*)$ and

$$Pos(G(d^*), I) \subseteq Pos(F, I) \subseteq \mathcal{H}.$$

By the induction hypothesis, it follows that $\langle \mathcal{H}, I\rangle \models_{ht} G(d^*)$. We can conclude that $\langle \mathcal{H}, I\rangle \models_{ht} \forall X\, G(X)$. *Case 2.6:* F is $\exists X\, G(X)$. Similar to Case 2.5. □

The definition of $G^{sp}_I(\Gamma)$ in Section 6 is restricted to the case when Γ is a completable set of sentences over σ^I. It can be generalized to arbitrary sets of sentences as follows. A *rule subformula* of a formula F is an occurrence of an implication in F that does not belong to the antecedent of any implication (Ferraris *et al*. 2011, Section 7.3; Lee and Meng 2011, Section 3.3). Let I be an interpretation of an arbitrary signature σ, and let Γ be a set of sentences over σ^I. All vertices of $G^{sp}_I(\Gamma)$ are elements of $I^{\downarrow}$. The graph has an edge from A to B iff, for some sentence $F_1 \to F_2$ obtained from a rule subformula of a member of Γ by substituting names d^* for its free variables, $A \in Pos(F_2, I)$ and $B \in Pos(F_1, I)$.

For any sentence F, $G^{sp}_I(F)$ stands for $G^{sp}_I(\{F\})$.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

Lemma 2
For any HT-interpretation $\langle \mathcal{H}, I\rangle$, any atom M in $I^{\downarrow} \setminus \mathcal{H}$, and any sentence F over σ^I, if

(i) for every edge (M, B) of $G^{sp}_I(F)$, $B \in \mathcal{H}$,
(ii) $M \in Pos(F, I)$, and
(iii) $\langle \mathcal{H}, I\rangle \models_{ht} F$,

then $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$.

Proof
By induction on the size of F. Sentence F is neither atomic nor $\bot$. Indeed, in that case F would be an atomic sentence of the form $p(\mathbf{t})$, where p is intensional, because, by (ii), $Pos(F, I)$ is non-empty. Then, from (iii), $p((\mathbf{t}^I)^*) \in \mathcal{H}$. On the other hand, $Pos(F, I)$ is $\{p((\mathbf{t}^I)^*)\}$, and from (ii) we conclude that $M = p((\mathbf{t}^I)^*)$. This contradicts the assumption that $M \in I^{\downarrow} \setminus \mathcal{H}$. Thus five cases are possible.

Case 1: F is $F_1 \wedge F_2$. From (iii) we can conclude that $\langle \mathcal{H}, I\rangle \models_{ht} F_i$ for $i = 1, 2$. It is sufficient to show that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F_i$; then the conclusion that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$ will follow. *Case 1.1:* $M \in Pos(F_i, I)$, so that formula F_i satisfies condition (ii). That formula satisfies condition (i) as well, because $G^{sp}_I(F_i)$ is a subgraph of $G^{sp}_I(F)$, and it satisfies condition (iii). So the conclusion $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F_i$ follows by the induction hypothesis. *Case 1.2:* $M \notin Pos(F_i, I)$. Since $Pos(F_i, I)$ is a subset of $I^{\downarrow}$, $Pos(F_i, I) \subseteq I^{\downarrow} \setminus \{M\}$. On the other hand, from the fact that $\langle \mathcal{H}, I\rangle \models_{ht} F_i$ we conclude, by Proposition 3(a), that $I \models F_i$. By Lemma 1, it follows that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F_i$.

Case 2: F is $F_1 \vee F_2$. From (iii) we can conclude that $\langle \mathcal{H}, I\rangle \models_{ht} F_i$ for $i = 1$ or $i = 2$. It is sufficient to show that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F_i$; then the conclusion that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$ will follow. The reasoning is the same as in Case 1.

Case 3: F is $F_1 \to F_2$. Then, $Pos(F, I) = Pos(F_2, I)$. By (ii), it follows that

$$M \in Pos(F_2, I). \tag{28}$$

On the other hand, F is a rule subformula of itself, so that for every atom B in $Pos(F_1, I)$, (M, B) is an edge of the graph $G^{sp}_I(F)$. By (i), it follows that every such atom B belongs to $\mathcal{H}$. Consequently

$$Pos(F_1, I) \subseteq \mathcal{H}. \tag{29}$$

Case 3.1: $\langle \mathcal{H}, I\rangle \models_{ht} F_2$, so that F_2 satisfies condition (iii). Since $G^{sp}_I(F_2)$ is a subgraph of $G^{sp}_I(F)$, F_2 satisfies condition (i) as well. By (28), F_2 satisfies condition (ii). Then, by the induction hypothesis, $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F_2$. Consequently $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$. *Case 3.2:* $\langle \mathcal{H}, I\rangle \not\models_{ht} F_2$. Then, in view of (iii), $\langle \mathcal{H}, I\rangle \not\models_{ht} F_1$. From this fact and formula (29) we can conclude, by Lemma 1, that $I \not\models F_1$. By Proposition 3(a), it follows that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \not\models_{ht} F_1$, which implies $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$.

Case 4: F is $\forall X\, G(X)$. From (iii) we can conclude that for every d in the domain $|I|^s$, where s is the sort of X, $\langle \mathcal{H}, I\rangle \models_{ht} G(d^*)$. It is sufficient to show that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} G(d^*)$; then the conclusion that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$ will follow. *Case 4.1:* $M \in Pos(G(d^*), I)$, so that formula $G(d^*)$ satisfies condition (ii). That formula satisfies condition (i) as well, because $G^{sp}_I(G(d^*))$ is a subgraph of $G^{sp}_I(F)$, and it satisfies condition (iii). So the conclusion $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} G(d^*)$ follows by the induction hypothesis. *Case 4.2:* $M \notin Pos(G(d^*), I)$. Since $Pos(G(d^*), I)$ is a subset of $I^{\downarrow}$, we can conclude that $Pos(G(d^*), I) \subseteq I^{\downarrow} \setminus \{M\}$. On the other hand, from the fact that

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

$\langle \mathcal{H}, I\rangle \models_{ht} G(d^*)$ we conclude, by Proposition 3(a), that $I \models G(d^*)$. By Lemma 1, it follows that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} G(d^*)$.

Case 5: F is $\exists X\, G(X)$. From (iii) we can conclude that for some d in the domain $|I|^s$, where s is the sort of X, $\langle \mathcal{H}, I\rangle \models_{ht} G(d^*)$. It is sufficient to show that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} G(d^*)$; then the conclusion that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$ will follow. The reasoning is the same as in Case 4. □

Lemma 3
For any interpretation I of σ and any set Γ of sentences over σ^I such that the graph $G^{sp}_I(\Gamma)$ has no infinite walks, I is a stable model of Γ iff I is pointwise stable.

Proof
We need to show that if I is a model of Γ such that the graph $G^{sp}_I(\Gamma)$ has no infinite walks, and there exists a proper subset $\mathcal{H}$ of $I^{\downarrow}$ such that $\langle \mathcal{H}, I\rangle$ satisfies Γ, then a subset with this property can be obtained from $I^{\downarrow}$ by removing a single element.

The set $I^{\downarrow} \setminus \mathcal{H}$ contains an atom M such that for every edge (M, B) of the graph $G^{sp}_I(\Gamma)$, $B \notin I^{\downarrow} \setminus \mathcal{H}$. Indeed, otherwise this graph would have an infinite walk consisting of elements of $I^{\downarrow} \setminus \mathcal{H}$. On the other hand, for every such edge, $B \in I^{\downarrow}$. Indeed, from the definition of the graph $G^{sp}_I(\Gamma)$ we see that for every edge (M, B) of that graph, B belongs to the set $Pos(F, I)$ for some sentence F, and that set is contained in $I^{\downarrow}$. Consequently for every edge (M, B) of $G^{sp}_I(\Gamma)$, $B \in \mathcal{H}$.

We will show that $\langle I^{\downarrow} \setminus \{M\}, I\rangle$ satisfies Γ. Take a sentence F from Γ. *Case 1:* $M \in Pos(F, I)$. Then, condition (ii) of Lemma 2 is satisfied for the HT-interpretation $\langle \mathcal{H}, I\rangle$. Condition (i) is satisfied for this HT-interpretation as well, because $G^{sp}_I(F)$ is a subgraph of $G^{sp}_I(\Gamma)$; furthermore, condition (iii) is satisfied because $\langle \mathcal{H}, I\rangle$ satisfies Γ. Consequently $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$ by Lemma 2. *Case 2:* $M \notin Pos(F, I)$. Then, $Pos(F, I) \subseteq I^{\downarrow} \setminus \{M\}$. Since $I \models F$, we can conclude that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$ by Lemma 1. □

A model I of a set Γ of completable sentences over σ is *supported* if for every atom $p(\mathbf{d}^*)$ in $I^{\downarrow}$ there exists an instance $F \to p(\mathbf{d}^*)$ of a member of Γ such that $I \models F$.

Lemma 4
Every pointwise stable model of a completable set of sentences is supported.

Proof
Let I be a pointwise stable model of a completable set Γ of sentences. Take an atom $p(\mathbf{d}^*)$ from $I^{\downarrow}$. We need to find an instance $F \to p(\mathbf{d}^*)$ of a completable sentence from Γ such that $I \models F$.

By the definition of a pointwise stable model, $\langle I^{\downarrow} \setminus \{p(\mathbf{d}^*)\}, I\rangle$ does not satisfy Γ. Then, one of the completable sentences from Γ has an instance $F \to G$ such that

$$\langle I^{\downarrow} \setminus \{p(\mathbf{d}^*)\}, I\rangle \not\models_{ht} F \to G. \tag{30}$$

We will show that this instance has the required properties. Since I is a model of Γ,

$$I \models F \to G. \tag{31}$$

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

From (30) and (31) we conclude that

$$\langle I^{\downarrow} \setminus \{p(\mathbf{d}^*)\}, I\rangle \models_{ht} F, \tag{32}$$

and

$$\langle I^{\downarrow} \setminus \{p(\mathbf{d}^*)\}, I\rangle \not\models_{ht} G. \tag{33}$$

From (32) and Proposition 3(a), $I \models F$. Then, in view of (31),

$$I \models G. \tag{34}$$

From (33), (34) and Proposition 3(c) we can conclude that formula $G\mathrm{d}$ contains p, so that it has the form $p(\mathbf{e}^*)$ for some tuple of domain elements $\mathbf{e}$. Then, from (34), $p(\mathbf{e}^*) \in I^{\downarrow}$, and from (33), $p(\mathbf{e}^*) \notin I^{\downarrow} \setminus \{p(\mathbf{d}^*)\}$. Consequently $p(\mathbf{e}^*)$ is $p(\mathbf{d}^*)$, so that $\mathbf{e} = \mathbf{d}$. □

Lemma 5
For any interpretation I of σ and any completable set Γ of sentences over σ such that the graph $G_I^{sp}(\Gamma)$ has no infinite walks, I is a supported model of Γ iff I is stable.

Proof
The if part follows from Lemmas 3 and 4. For the only if part, consider a supported model I of a completable set Γ of sentences such that the graph $G_I^{sp}(\Gamma)$ has no infinite walks; we need to prove that I is stable. According to Lemma 3, it is sufficient to check that I is pointwise stable.

Take any atom M in $I^{\downarrow}$; we will show that $\langle I^{\downarrow} \setminus \{M\}, I\rangle$ is not an HT-model of Γ. Since I is supported, one of the completable sentences in Γ has an instance $F \to M$ such that $I \models F$. Atom M does not belong to $Pos(F, I)$, because otherwise $M, M, \ldots$ would be an infinite walk in $G_I^{sp}(\Gamma)$. Since the set $Pos(F, I)$ is a subset of $I^{\downarrow}$, we can conclude that it is a subset of $I^{\downarrow} \setminus \{M\}$. By Lemma 1, it follows that $\langle I^{\downarrow} \setminus \{M\}, I\rangle \models_{ht} F$. Therefore $\langle I^{\downarrow} \setminus \{M\}, I\rangle \not\models_{ht} F \to M$. □

Lemma 6
For any interpretation I of σ and any completable set Γ sentences over σ, I is a supported model of Γ iff $I \models \mathrm{COMP}[\Gamma]$.

Proof
Let the sentences defining an intensional predicate symbol p in Γ be

$$\forall \mathbf{U}_i \mathbf{V}(F_i(\mathbf{U}_i, \mathbf{V}) \to p(\mathbf{V})) \qquad (i = 1, \ldots, k). \tag{35}$$

The completed definition of p is

$$\forall \mathbf{V} \left(p(\mathbf{V}) \leftrightarrow \bigvee_{i=1}^{k} \exists \mathbf{U}_i \, F_i(\mathbf{U}_i, \mathbf{V}) \right).$$

Hence, an interpretation I of σ satisfies $\mathrm{COMP}[\Gamma]$ iff the following three conditions hold:

(a) for every intensional p, I satisfies the sentence

$$\forall \mathbf{V} \left(p(\mathbf{V}) \to \bigvee_{i=1}^{k} \exists \mathbf{U}_i \, F_i(\mathbf{U}_i, \mathbf{V}) \right);$$

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

(b) for every intensional p and for every i, I satisfies the sentence

$$\forall \mathbf{V}(\exists \mathbf{U}_i\, F_i(\mathbf{U}_i, \mathbf{V}) \to p(\mathbf{V})); \tag{36}$$

(c) I satisfies all constraints of Γ.

Since (36) is equivalent to (35), I is a model of Γ if and only if conditions (b) and (c) hold. It remains to check that condition (a) holds if and only if the model I is supported.

Condition (a) can be expressed by saying that

$$\text{for every atom } p(\mathbf{d}^*) \text{ in } I^{\downarrow} \text{ there exists } i \text{ such that } I \models \exists \mathbf{U}_i\, F_i(\mathbf{U}_i, \mathbf{d}^*),$$

or, equivalently, that

$$\text{for every atom } p(\mathbf{d}^*) \text{ in } I^{\downarrow} \text{ there exist } i \text{ and a tuple } \mathbf{d}_i \text{ of domain elements such that } I \models F_i(\mathbf{d}_i^*, \mathbf{d}^*).$$

Since the members of Γ defining p are sentences of form (35), the last condition expresses that I is a supported model of Γ. □

The assertion of the Main Lemma follows from Lemmas 5 and 6.

8 Proof of Theorem 1

The proof of Theorem 1 refers to infinitary propositional formulas and the strong equivalence relation between them (Harrison *et al.* 2017).

Translations τ and τ^* are related by a third translation $F \mapsto F^{\text{prop}}$ (Lifschitz *et al.* 2019, Section 5), which transforms sentences over σ_0 into infinitary propositional combinations of precomputed atoms. This translation is defined as follows:

- if F is $p(t_1, \dots, t_n)$, then F^{prop} is obtained from F by replacing each t_i by the value obtained after evaluating all arithmetic functions in t_i;
- if F is $t_1\, rel\, t_2$, then F^{prop} is $\top$ if the values of t_1 and t_2 are in the relation rel, and $\bot$ otherwise;
- $\bot^{\text{prop}}$ is $\bot$;
- $(F \odot G)^{\text{prop}}$ is $F^{\text{prop}} \odot G^{\text{prop}}$ for every binary connective $\odot$;
- $(\forall X\, F(X))^{\text{prop}}$ is the conjunction of the formulas $F(r)^{\text{prop}}$ over all precomputed terms r if X is a general variable, and over all numerals r if X is an integer variable;
- $(\exists X\, F(X))^{\text{prop}}$ is the disjunction of the formulas $F(r)^{\text{prop}}$ over all precomputed terms r if X is a general variable, and over all numerals r if X is an integer variable.

This translation is similar to the grounding operation defined by Truszczynski (2012, Section 2). The following proposition, analogous to Proposition 2 from Truszczynski's paper and to Proposition 3 by Lifschitz *et al.* (2019), relates the meaning of a sentence to the meaning of its propositional translation. It differs from the last result in view of the fact that the division and modulo operations are treated here in a different way (see Section 2.1.2 and Footnote 2), but can be proved in a similar way.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

Proposition 2
For any rule R, $(\tau^* R)^{\text{prop}}$ is strongly equivalent to τR.

Since standard interpretations of σ_0 are semi-Herbrand (Section 6), the correspondence between tuples $\mathbf{d}$ of elements of domains of a standard interpretation and tuples $\mathbf{d}^*$ of their names is one-to-one, and we take the liberty to identify them. Therefore, for a standard interpretation I of σ_0, $I^{\downarrow}$ is identified with the set of precomputed atoms that are satisfied by I. In view of this convention, the transformation $I \mapsto I^{\downarrow}$ is the inverse of the transformation $\mathcal{J} \mapsto \mathcal{J}^{\uparrow}$, defined in Section 2.3: for any set $\mathcal{J}$ of precomputed atoms over σ_0,

$$(\mathcal{J}^{\uparrow})^{\downarrow} = \mathcal{J}. \tag{37}$$

Lemma 7 (*Fandinno and Lifschitz 2023, Lemma 2(i)*)
A standard interpretation I of σ_0 satisfies a sentence F over σ_0 iff $I^{\downarrow}$ satisfies F^{prop}.

The lemma below relates the meaning of a sentence in the logic of here-and-there to the meaning of its grounding in the infinitary version of that logic (Truszczynski 2012, Section 2). It is similar to Proposition 4 from that paper and can be proved by induction in a similar way.

Lemma 8
A standard HT-interpretation $\langle \mathcal{H}, I \rangle$ of σ_0 satisfies a sentence F over σ_0 iff $\langle \mathcal{H}, I^{\downarrow} \rangle$ satisfies F^{prop}.

The following lemma relates stable models of first-order formulas (as defined in Appendix B) to stable models of infinitary propositional formulas (Truszczynski 2012, Section 2). It is a generalization of Theorem 5 from that paper.

Lemma 9
A standard interpretation I of σ_0 is a stable model of a set Γ of sentences over σ_0 iff $I^{\downarrow}$ is a stable model of $\{F^{\text{prop}} : F \in \Gamma\}$.

Proof
By Lemma 7, an interpretation I is a model of Γ iff $I^{\downarrow}$ is a model of $\{F^{\text{prop}} : F \in \Gamma\}$. By Lemma 8, for any proper subset $\mathcal{H}$ of $I^{\downarrow}$, $\langle \mathcal{H}, I \rangle$ satisfies Γ iff $\langle \mathcal{H}, I^{\downarrow} \rangle$ satisfies $\{F^{\text{prop}} : F \in \Gamma\}$. □

Proof of Theorem 1
For any set $\mathcal{J}$ of precomputed atoms and any mini-GRINGO program Π,

$\mathcal{J}^{\uparrow}$ is a stable model of $\tau^*\Pi$
iff
$(\mathcal{J}^{\uparrow})^{\downarrow}$ is a stable model of $\{F^{\text{prop}} : F \in \tau^*\Pi\}$ (Lemma 9)
iff
$\mathcal{J}$ is a stable model of $\{F^{\text{prop}} : F \in \tau^*\Pi\}$ (formula (37))
iff
$\mathcal{J}$ is a stable model of $\{(\tau^* R)^{\text{prop}} : R \in \Pi\}$ (definition of $\tau^*\Pi$)
iff
$\mathcal{J}$ is a stable model of $\{\tau R : R \in \Pi\}$ (Proposition 2)

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

iff

$\mathcal{J}$ is a stable model of Π (semantics of mini-GRINGO). □

9 Proof of Theorem 2

The equivalence between conditions (b) and (c) in the statement of Theorem 2 follows from the fact that for every sentence F over σ_0 (first-order or second-order), $\mathcal{P}^{\uparrow}$ satisfies $v(F)$ iff $\mathcal{P}^v$ satisfies F. The equivalence between conditions (a) and (b) will be derived from the Main Lemma. To this end, we need to relate the positive dependency graph of an io-program to the positive dependency graph of the corresponding completable set of sentences with respect to a standard interpretation of σ_0. Such a relationship is described by Lemma 14 below.

Lemma 10
For any tuple $\mathbf{t}$ of ground terms in the language of mini-GRINGO and for any tuple $\mathbf{r}$ of precomputed terms of the same length, the formula $val_{\mathbf{t}}(\mathbf{r})$ is equivalent to $\top$ if $\mathbf{r} \in [\mathbf{t}]$, and to $\bot$ otherwise.

Proof
The special case when $\mathbf{t}$ is a single term is Lemma 1 by Lifschitz et al. (2020). The general case easily follows. □

The following lemma describes properties of the transformation τ^B, defined in Section 2.4.

Lemma 11
For any set $\mathcal{J}$ of precomputed atoms and any ground literal L such that $\mathcal{J}^{\uparrow} \models \tau^B(L)$,

(a) If L is $p(\mathbf{t})$ or *not not* $p(\mathbf{t})$ then, for some tuple $\mathbf{r}$ in $[\mathbf{t}]$, $p(\mathbf{r}) \in \mathcal{J}$.
(b) If L is *not* $p(\mathbf{t})$ then, for some tuple $\mathbf{r}$ in $[\mathbf{t}]$, $p(\mathbf{r}) \notin \mathcal{J}$.

Proof
Consider the case when L is $p(\mathbf{t})$. Then, $\tau^B(L)$ is $\exists \mathbf{V}(val_{\mathbf{t}}(\mathbf{V}) \wedge p(\mathbf{V}))$. Since $\mathcal{J}^{\uparrow}$ satisfies this formula, there exists a tuple $\mathbf{r}$ of precomputed terms such that $\mathcal{J}^{\uparrow}$ satisfies $val_{\mathbf{t}}(\mathbf{r})$ and $p(\mathbf{r})$. Since $val_{\mathbf{t}}(\mathbf{r})$ is satisfiable, we can conclude by Lemma 10 that $\mathbf{r} \in [\mathbf{t}]$. The other two cases are analogous. □

Lemma 12
If $\mathbf{U}$ is a tuple of general variables, and $\mathbf{u}$ is a tuple of precomputed terms of the same length, then

(a) for any term $t(\mathbf{U})$ and any precomputed term r, the result of substituting $\mathbf{u}$ for $\mathbf{U}$ in $val_{t(\mathbf{U})}(r)$ is $val_{t(\mathbf{u})}(r)$;
(b) for any conjunction $Body(\mathbf{U})$ of literals and comparisons, the result of substituting $\mathbf{u}$ for $\mathbf{U}$ in $\tau^B(Body(\mathbf{U}))$ is $\tau^B(Body(\mathbf{u}))$.

Proof
Part (i) is easy to prove by induction. Part (ii) immediately follows. □

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

In the rest of this section, v is a valuation on the set of placeholders of an io-program Ω, and $\mathcal{J}$ is a set of precomputed atoms that do not contain placeholders of Ω. Standard interpretations of σ_0 are semi-Herbrand, and the references to Pos and G^{sp} below refer to the definitions modified as described at the end of Section 6.

Lemma 13
Let $\mathbf{U}$ be a list of distinct general variables, and let $p(\mathbf{t}(\mathbf{U})) \leftarrow Body(\mathbf{U})$ be a basic rule of Ω with all variables explicitly shown. For any tuple $\mathbf{u}$ of precomputed terms of the same length as $\mathbf{U}$, any tuple $\mathbf{r}$ from $[v(\mathbf{t}(\mathbf{u}))]$, and any precomputed atom A from $Pos(\tau^B(v(Body(\mathbf{u}))), \mathcal{J}^\uparrow)$, the positive dependency graph of Ω for the input $(v, \mathcal{J}^{in})$ has an edge from $p(\mathbf{r})$ to A.

Proof
We will show that the instance

$$p(v(\mathbf{t}(\mathbf{u}))) \leftarrow v(Body(\mathbf{u})),$$

of the rule

$$p(v(\mathbf{t}(\mathbf{U}))) \leftarrow v(Body(\mathbf{U})),$$

satisfies conditions (a)–(e) imposed on R in the definition of the positive dependency graph of an io-program (see Section 4).

Verification of condition (a): the argument of p in the head of R is $v(\mathbf{t}(\mathbf{u}))$, and $\mathbf{r} \in [v(\mathbf{t}(\mathbf{u}))]$.

Verification of condition (b): since $A \in Pos(\tau^B(v(Body(\mathbf{u}))), \mathcal{J}^\uparrow)$, the body $v(Body(\mathbf{u}))$ of R has a conjunctive term $p'(\mathbf{r}')$ such that

$$A \in Pos(\tau^B(p'(\mathbf{r}')), \mathcal{J}^\uparrow) = Pos(\exists \mathbf{V}(\mathbf{r}' = \mathbf{V} \wedge p'(\mathbf{V})), \mathcal{J}^\uparrow) = Pos(p'(\mathbf{r}')), \mathcal{J}^\uparrow) \subseteq \{p'(\mathbf{r}')\}.$$

It follows that $A = p'(\mathbf{r}')$, so that A is a conjunctive term of the body required by condition (b).

Verification of conditions (c)–(e): let L be a conjunctive term of the body $v(Body(\mathbf{u}))$ of R that contains an input symbol of Ω or is a comparison. Since $Pos(\tau^B(v(Body(\mathbf{u}))), \mathcal{J}^\uparrow)$ is non-empty, $\tau^B(v(Body(\mathbf{u})))$ is satisfied by $\mathcal{J}^\uparrow$. Therefore, the conjunctive term $\tau^B(L)$ of that formula is satisfied by $\mathcal{J}^\uparrow$ as well. If L has the form $q(\mathbf{t})$ or *not not* $q(\mathbf{t})$ then, by Lemma 11(a), there exists a tuple $\mathbf{r}'$ in $[\mathbf{t}]$ such that $q(\mathbf{r}') \in \mathcal{J}$, and consequently $q(\mathbf{r}') \in \mathcal{J}^{in}$. If L has the form *not* $q(\mathbf{t})$ then, by Lemma 11(b), there exists a tuple $\mathbf{r}'$ in $[\mathbf{t}]$ such that $q(\mathbf{r}') \notin \mathcal{J}$, and consequently $q(\mathbf{r}') \notin \mathcal{J}^{in}$. If L is a comparison $t_1 \prec t_2$ then $\tau^B(L)$ is

$$\exists V_1 V_2(val_{t_1}(V_1) \wedge val_{t_2}(V_2) \wedge V_1 \prec V_2).$$

Since this formula is satisfied by $\mathcal{J}^\uparrow$, the relation $\prec$ holds for a pair (r_1, r_2) of precomputed terms such that $val_{t_1}(r_1)$ and $val_{t_2}(r_2)$ are satisfied by $\mathcal{J}^\uparrow$. By Lemma 10, $r_1 \in [t_1]$ and $r_2 \in [t_2]$. □

Lemma 14
Let Π be the set of rules of Ω.

(i) The positive dependency graph of Ω for the input $(v, \mathcal{J}^{in})$ is a supergraph of the positive dependency graph of $\tau^*(v(\Pi))$.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

(ii) If Ω is locally tight for the input $(v, \mathcal{J}^{in})$ then the graph $G^{sp}_{\mathcal{J}^\uparrow}(\tau^*(v(\Pi)))$ has no infinite walks.

Proof
(i) Replacing a choice rule $\{p(\mathbf{t})\} \leftarrow Body$ in Ω by the basic rule $p(\mathbf{t}) \leftarrow Body$ affects neither the positive dependency graph of Ω nor the positive dependency graph of $\tau^*(v(\Pi))$. Consequently we can assume, without loss of generality, that Ω is an io-program without choice rules.

Pick any edge $(p(\mathbf{r}), A)$ of the (modified) graph $G^{sp}_{\mathcal{J}^\uparrow}(\tau^*(v(\Pi)))$. Then, there is an instance $F \to G$ of the completed sentence obtained by applying τ^* to a basic rule of $v(\Pi)$ such that $p(\mathbf{r}) \in Pos(G, \mathcal{J}^\uparrow)$ and $A \in Pos(F, \mathcal{J}^\uparrow)$. That rule of $v(\Pi)$ can be written as

$$p(v(\mathbf{t}(\mathbf{U}))) \leftarrow v(Body(\mathbf{U})), \tag{38}$$

where

$$p(\mathbf{t}(\mathbf{U})) \leftarrow Body(\mathbf{U}),$$

is a rule of Π, and $\mathbf{U}$ is the list of its variables. The result of applying τ^* to (38) is

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

where $\mathbf{V}$ is a tuple of general variables. Let $\mathbf{u}$, $\mathbf{z}$ be tuples of precomputed terms that are substituted for $\mathbf{U}$, $\mathbf{V}$ in the process of forming the instance $F \to G$. By Lemma 12, that instance can be written as

$$val_{v(\mathbf{t}(\mathbf{u}))}(\mathbf{v}) \wedge \tau^B(v(Body(\mathbf{u}))) \to p(\mathbf{v}).$$

Consequently $p(\mathbf{r}) \in Pos(p(\mathbf{v}), \mathcal{J}^\uparrow)$ and

$$A \in Pos(val_{v(\mathbf{t}(\mathbf{u}))}(\mathbf{v}) \wedge \tau^B(v(Body(\mathbf{u}))), \mathcal{J}^\uparrow).$$

The first of these conditions implies $\mathbf{v} = \mathbf{r}$, so that the second can be rewritten as

$$A \in Pos(val_{v(\mathbf{t}(\mathbf{u}))}(\mathbf{r}) \wedge \tau^B(v(Body(\mathbf{u}))), \mathcal{J}^\uparrow). \tag{39}$$

It follows that $\mathcal{J}^\uparrow$ satisfies $val_{v(\mathbf{t}(\mathbf{u}))}(\mathbf{r})$. By Lemma 10, we can conclude that $\mathbf{r} \in v(\mathbf{t}(\mathbf{u}))$. On the other hand, the set $Pos(val_{v(\mathbf{t}(\mathbf{u}))}(\mathbf{r}), \mathcal{J}^\uparrow)$ is empty, because the formula $val_{v(\mathbf{t}(\mathbf{u}))}(\mathbf{r})$ does not contain intensional symbols. Consequently (39) implies that

$$A \in Pos(\tau^B(v(Body(\mathbf{u}))), \mathcal{J}^\uparrow).$$

By Lemma 13, it follows that $(p(\mathbf{r}), A)$ is an edge of the positive dependency graph of Ω for the input $(v, \mathcal{J}^{in})$.

(ii) Immediate from part (i). □

Now we are ready to prove that condition (a) in the statement of Theorem 2 is equivalent to condition (b). The assumption that $\mathcal{P}$ is an io-model of Ω for an input $(v, \mathcal{I})$ implies that $\mathcal{P}^{in} = \mathcal{I}$. Indeed, that assumption means that $\mathcal{P}$ is the set of public atoms of some stable model $\mathcal{J}$ of the program $v(\Pi) \cup \mathcal{I}$, where Π is the set of rules of Ω. In that program, the facts $\mathcal{I}$ are the only rules containing input symbols in the head, so that $\mathcal{J}^{in} = \mathcal{I}$. Since all input symbols are public, it follows that $\mathcal{P}^{in} = \mathcal{I}$. It remains to

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

show that

$$\mathcal{P} \text{ is an io-model of } \Omega \text{ for } (v, \mathcal{P}^{in}) \text{ iff } \mathcal{P}^{\uparrow} \text{ satisfies } v(\mathrm{COMP}[\Omega]), \tag{40}$$

under the assumption that Ω is locally tight for $(v, \mathcal{P}^{in})$.

Recall that $\mathrm{COMP}[\Omega]$ is the second-order sentence $\exists P_1 \cdots P_l\, C$, where C is obtained from the sentence $\mathrm{COMP}[\tau^*\Pi]$ by replacing the private symbols $p_1/n_1, \ldots, p_l/n_l$ of Ω with distinct predicate variables $P_1, \ldots, P_l$. Then $v(\mathrm{COMP}[\Omega])$ is $\exists P_1 \cdots P_l\, v(C)$. Hence, the condition in the right-hand side of (40) means that

> for some set $\mathcal{J}$ obtained from $\mathcal{P}$ by adding private precomputed atoms,
> $\mathcal{J}^{\uparrow}$ satisfies $v(\mathrm{COMP}[\tau^*\Pi])$.

Note also that $v(\mathrm{COMP}[\tau^*\Pi]) = \mathrm{COMP}[\tau^*(v(\Pi)]$. Thus the right-hand side of (40) is equivalent to the condition

> for some set $\mathcal{J}$ obtained from $\mathcal{P}$ by adding private precomputed atoms,
> $\mathcal{J}^{\uparrow}$ satisfies $\mathrm{COMP}[\tau^*(v(\Pi))]$. (41)

The condition in the left-hand side of (40) means that $\mathcal{P}$ is the set of public atoms of some stable model of $v(\Pi) \cup \mathcal{P}^{in}$. By Theorem 1, this condition can be equivalently reformulated as follows:

> for some set $\mathcal{J}$ obtained from $\mathcal{P}$ by adding private precomputed atoms,
> $\mathcal{J}^{\uparrow}$ is a stable model of $\tau^*(v(\Pi) \cup \mathcal{P}^{in})$. (42)

We can further reformulate this condition using the Main Lemma (Section 6) with

- the signature σ_0 as σ,
- all comparison symbols viewed as extensional,
- $\mathcal{J}^{\uparrow}$ as I, and
- $\tau^*(v(\Pi) \cup \mathcal{P}^{in})$ as Γ.

The graph $G^{sp}_{\mathcal{J}^{\uparrow}}(\tau^*(v(\Pi) \cup \mathcal{P}^{in}))$ has no infinite walks, because it is identical to the graph $G^{sp}_{\mathcal{J}^{\uparrow}}(\tau^*(v(\Pi)))$, which has no infinite walks by Lemma 14(ii). Consequently (42) – and so the left-hand side of (40) – is equivalent to the condition:

> for some set $\mathcal{J}$ obtained from $\mathcal{P}$ by adding private precomputed atoms,
> $\mathcal{J}^{\uparrow}$ satisfies $\mathrm{COMP}[\tau^*(v(\Pi)) \cup \mathcal{P}^{in}]$. (43)

Therefore, condition (40) is equivalent to the assertion that conditions (41) and (43) are equivalent to each other. So to prove (40), it is enough to show that

> for every set $\mathcal{J}$ obtained from $\mathcal{P}$ by adding private precomputed atoms,
> $\mathcal{J}^{\uparrow}$ satisfies $\mathrm{COMP}[\tau^*(v(\Pi))]$ iff $\mathcal{J}^{\uparrow}$ satisfies $\mathrm{COMP}[\tau^*(v(\Pi)) \cup \mathcal{P}^{in}]$. (44)

Formula $\mathrm{COMP}[\tau^*(v(\Pi)) \cup \mathcal{P}^{in}]$ is a conjunction that includes the completed definitions of all input symbols among its conjunctive terms. The interpretation $\mathcal{J}^{\uparrow}$ satisfies these completed definitions, because $\mathcal{J}^{in} = \mathcal{P}^{in}$. On the other hand, the remaining conjunctive terms of $\mathrm{COMP}[\tau^*(v(\Pi)) \cup \mathcal{P}^{in}]$ form the completion $\mathrm{COMP}[\tau^*(v(\Pi))]$ under the assumption that both comparison symbols and the input symbols are considered extensional. Hence, (44) holds.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

10 Proof of Theorem 3

IO-programs Ω and Ω' are equivalent to each other under an assumption A iff, for every input $(v, \mathcal{I})$ from the domain defined by A and every set $\mathcal{P}$ of precomputed public atoms,

$$\mathcal{P} \text{ is an io-model of } \Omega \text{ for } (v, \mathcal{I}) \text{ iff } \mathcal{P} \text{ is an io-model of } \Omega' \text{ for } (v, \mathcal{I}).$$

Since Ω and Ω' are locally tight under assumption A, they both are locally tight for input $(v, \mathcal{I})$. Hence, by Theorem 2, the above condition is equivalent to:

$$\mathcal{P}^v \text{ satisfies COMP}[\Omega] \text{ and } \mathcal{P}^{in} = \mathcal{I} \quad \text{iff} \quad \mathcal{P}^v \text{ satisfies COMP}[\Omega'] \text{ and } \mathcal{P}^{in} = \mathcal{I}.$$

We can further rewrite this condition as

$$\mathcal{P}^{in} = \mathcal{I} \text{ implies that } \mathcal{P}^v \text{ satisfies COMP}[\Omega] \leftrightarrow \text{COMP}[\Omega'].$$

Hence, programs Ω and Ω' are equivalent to each other under assumption A iff the condition

$$\mathcal{P}^v \text{ satisfies COMP}[\Omega] \leftrightarrow \text{COMP}[\Omega'], \tag{45}$$

holds for every input $(v, \mathcal{I})$ from the domain defined by A and every set $\mathcal{P}$ of precomputed public atoms such that $\mathcal{P}^{in} = \mathcal{I}$; equivalently, iff

$$\begin{array}{c}(45) \text{ holds for every input } (v, \mathcal{I}) \text{ such that } \mathcal{I}^v \text{ satisfies } A \\ \text{and every set } \mathcal{P} \text{ of precomputed public atoms} \\ \text{such that } \mathcal{P}^{in} = \mathcal{I};\end{array}$$

equivalently, iff

$$\begin{array}{c}(45) \text{ holds for every input } (v, \mathcal{I}) \\ \text{and every set } \mathcal{P} \text{ of precomputed public atoms} \\ \text{such that } \mathcal{P}^{in} = \mathcal{I} \text{ and } \mathcal{P}^v \text{ satisfies } A;\end{array}$$

equivalently, iff

$$\begin{array}{c}(45) \text{ holds for every valuation } v \text{ of } PH \\ \text{and every set } \mathcal{P} \text{ of precomputed public atoms such that } \mathcal{P}^v \text{ satisfies } A;\end{array}$$

equivalently, iff for every valuation v of PH and every set $\mathcal{P}$ of precomputed public atoms, $\mathcal{P}^v$ satisfies the sentence

$$A \rightarrow (\text{COMP}[\Omega] \leftrightarrow \text{COMP}[\Omega']).$$

It remains to observe that an interpretation of σ_0 is standard for PH iff it can be represented in the form $\mathcal{P}^v$ for some $\mathcal{P}$ and v.

11 Conclusion

In this paper, we defined when an io-program is considered locally tight for an input, and proved two properties of locally tight programs: io-models can be characterized in terms of completion (Theorem 2), and the ANTHEM-P2P algorithm can be used to verify equivalence (Theorem 3). The local tightness condition is satisfied for many nontight programs that describe dynamic domains.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

Theorem 2 can be used also to justify using the ANTHEM algorithm (Fandinno *et al.* 2020) for verifying a locally tight program with respect to a specification expressed by first-order sentences.

Future work will include extending the ANTHEM-P2P algorithm and Theorem 3 to ASP programs that involve aggregate expressions, using the ideas of Fandinno et al. (2022) and Lifschitz (2022).

References

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. 1994. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science 1*, 51–60.

Fandinno, J., Hansen, Z. and Lierler, Y. 2022. Axiomatization of aggregates in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Fandinno, J., Hansen, Z., Lierler, Y., Lifschitz, V. and Temple, N. 2023. External behavior of a logic program and verification of refactoring. *Theory and Practice of Logic Programming*.

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*. To appear, URL: https://www.cs.utexas.edu/users/vl/papers/omega.pdf.

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*.

Ferraris, P. 2005. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 119–131.

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. URL: https://github.com/potassco/guide/releases/.

Gebser, M., Kaufmann, B., Neumann, A. and Schaub, T. 2007. clasp: A conflict-driven answer set solver. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*.

Gebser, M., Schaub, T. and Thiele, S. 2007. Gringo: A new grounder for answer set programming. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning*, 266–271.

Harrison, A., Lifschitz, V., Pearce, D. and Valverde, A. 2017. Infinitary equilibrium logic and strongly equivalent logic programs. *Artificial Intelligence 246*, 22–33.

Kovaćs, L. and Voronkov, A. 2013. First-order theorem proving and Vampire. In *International Conference on Computer Aided Verification*, 1–35.

Lee, J. and Meng, Y. 2011. First-order stable model semantics and first-order loop formulas. *Journal of Artificial Inteligence Research (JAIR) 42*, 125–180.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

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

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

Lifschitz, V. 2022. Strong equivalence of logic programs with counting. *Theory and Practice of Logic Programming 22*.

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., Lühne, P. and Schaub, T. 2020. Towards verifying logic programs in the input language of clingo. In *Fields of Logic and Computation III, Essays Dedicated to Yuri Gurevich on the Occasion of His 80th Birthday*. Springer, 190–209.

Lifschitz, V., Morgenstern, L. and Plaisted, D. 2008. Knowledge representation and classical logic. In *Handbook of Knowledge Representation*, F. van Harmelen, V. Lifschitz, and B. Porter, Eds. Elsevier, 3–88.

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

Pearce, D. and Valverde, A. 2004. Towards a first order equilibrium logic for nonmonotonic reasoning. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, 147–160.

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

Appendix A Many-sorted formulas

A many-sorted signature consists of symbols of three kinds – *sorts*, *function constants*, and *predicate constants*. A reflexive and transitive *subsort* relation is defined on the set of sorts. A tuple $s_1, \dots, s_n$ $(n \geq 0)$ of *argument sorts* is assigned to every function constant and to every predicate constant; in addition, a *value sort* is assigned to every function constant. Function constants with $n = 0$ are called *object constants*.

We assume that for every sort, an infinite sequence of *object variables* of that sort is chosen. *Terms* over a many-sorted signature σ are defined recursively:

- object constants and object variables of a sort s are terms of sort s;
- if f is a function constant with argument sorts $s_1, \dots, s_n$ $(n > 0)$ and value sort s, and $t_1, \dots, t_n$ are terms such that the sort of t_i is a subsort of s_i $(i = 1, \dots, n)$, then $f(t_1, \dots, t_n)$ is a term of sort s.

Atomic formulas over σ are

- expressions of the form $p(t_1, \dots, t_n)$, where p is a predicate constant and $t_1, \dots, t_n$ are terms such that their sorts are subsorts of the argument sorts $s_1, \dots, s_n$ of p, and
- expressions of the form $t_1 = t_2$, where t_1 and t_2 are terms such that their sorts have a common supersort.

First-order formulas over σ are formed from atomic formulas and the 0-place connective $\bot$ (falsity) using the binary connectives $\wedge$, $\vee$, $\rightarrow$ and the quantifiers $\forall$, $\exists$. The other connectives are treated as abbreviations: $\neg F$ stands for $F \rightarrow \bot$ and $F \leftrightarrow G$ stands for

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

$(F \to G) \wedge (G \to F)$. *First-order sentences* are first-order formulas without free variables. The *universal closure* $\widetilde{\forall} F$ of a formula F is the sentence $\forall \mathbf{X}\, F$, where $\mathbf{X}$ is the list of free variables of F.

An *interpretation* I of a signature σ assigns

- a non-empty *domain* $|I|^s$ to every sort s of I, so that $|I|^{s_1} \subseteq |I|^{s_2}$ whenever s_1 is a subsort of s_2,
- a function f^I from $|I|^{s_1} \times \cdots \times |I|^{s_n}$ to $|I|^s$ to every function constant f with argument sorts $s_1, \dots, s_n$ and value sort s, and
- a Boolean-valued function p^I on $|I|^{s_1} \times \cdots \times |I|^{s_n}$ to every predicate constant p with argument sorts $s_1, \dots, s_n$.

If I is an interpretation of a signature σ then by σ^I we denote the signature obtained from σ by adding, for every element d of a domain $|I|^s$, its *name* d^* as an object constant of sort s. The interpretation I is extended to σ^I by defining $(d^*)^I = d$. The value t^I assigned by an interpretation I of σ to a ground term t over σ^I and the satisfaction relation between an interpretation of σ and a sentence over σ^I are defined recursively, in the usual way. A *model* of a set Γ of sentences is an interpretation that satisfies all members of Γ.

If $\mathbf{d}$ is a tuple $d_1, \dots, d_n$ of elements of domains of I then $\mathbf{d}^*$ stands for the tuple $d_1^*, \dots, d_n^*$ of their names. If $\mathbf{t}$ is a tuple $t_1, \dots, t_n$ of ground terms then $\mathbf{t}^I$ stands for the tuple $t_1^I, \dots, t_n^I$ of values assigned to them by I.

Appendix B Stable models of many-sorted theories

The definition of a stable model given below is based on the first-order logic of here-and-there, which was introduced by Pearce and Valverde (2004) and Ferraris *et al.* (2011), and then extended to many-sorted formulas (Fandinno and Lifschitz 2023).

Consider a many-sorted signature σ with its predicate constants partitioned into *intensional* and *extensional*. For any interpretation I of σ, $I^\downarrow$ stands for the set of atoms of the form $p(\mathbf{d}^*)$ with intensional p that are satisfied by this interpretation.[8]

An *HT-interpretation* of σ is a pair $\langle \mathcal{H}, I \rangle$, where I is an interpretation of σ, and $\mathcal{H}$ is a subset of $I^\downarrow$. (In terms of Kripke models with two sorts, I is the there-world, and $\mathcal{H}$ describes the intensional predicates in the here-world). The satisfaction relation $\models_{ht}$ between HT-interpretation $\langle \mathcal{H}, I \rangle$ of σ and a sentence F over σ^I is defined recursively as follows:

- $\langle \mathcal{H}, I \rangle \models_{ht} p(\mathbf{t})$, where p is intensional, if $p((\mathbf{t}^I)^*) \in \mathcal{H}$;
- $\langle \mathcal{H}, I \rangle \models_{ht} p(\mathbf{t})$, where p is extensional, if $I \models p(\mathbf{t})$;
- $\langle \mathcal{H}, I \rangle \models_{ht} t_1 = t_2$ if $t_1^I = t_2^I$;
- $\langle \mathcal{H}, I \rangle \not\models_{ht} \bot$;
- $\langle \mathcal{H}, I \rangle \models_{ht} F \wedge G$ if $\langle \mathcal{H}, I \rangle \models_{ht} F$ and $\langle \mathcal{H}, I \rangle \models_{ht} G$;
- $\langle \mathcal{H}, I \rangle \models_{ht} F \vee G$ if $\langle \mathcal{H}, I \rangle \models_{ht} F$ or $\langle \mathcal{H}, I \rangle \models_{ht} G$;
- $\langle \mathcal{H}, I \rangle \models_{ht} F \to G$ if

[8] In earlier publications, this set of atoms was denoted by I^{int}. The symbol $I^\downarrow$ is more appropriate, because this operation is opposite to the operation $\mathcal{J}^\uparrow$, as discussed in Section 8.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press

(i) $\langle \mathcal{H}, I\rangle \not\models_{ht} F$ or $\langle \mathcal{H}, I\rangle \models_{ht} G$, and
(ii) $I \models F \to G$;

- $\langle \mathcal{H}, I\rangle \models_{ht} \forall X\, F(X)$ if $\langle \mathcal{H}, I\rangle \models_{ht} F(d^*)$ for each $d \in |I|^s$, where s is the sort of X;
- $\langle \mathcal{H}, I\rangle \models_{ht} \exists X\, F(X)$ if $\langle \mathcal{H}, I\rangle \models_{ht} F(d^*)$ for some $d \in |I|^s$, where s is the sort of X.

If $\langle \mathcal{H}, I\rangle \models_{ht} F$ holds, we say that $\langle \mathcal{H}, I\rangle$ *satisfies* F and that $\langle \mathcal{H}, I\rangle$ is an *HT-model* of F. If two formulas have the same HT-models then we say that they are *HT-equivalent.*

In the following proposition, we collected some properties of this satisfaction relation that can be proved by induction.

Proposition 3

(a) If $\langle \mathcal{H}, I\rangle \models_{ht} F$ then $I \models F$.
(b) For any sentence F that does not contain intensional symbols, $\langle \mathcal{H}, I\rangle \models_{ht} F$ iff $I \models F$.
(c) For any subset $\mathcal{S}$ of $\mathcal{H}$ such that the predicate symbols of its members do not occur in F, $\langle \mathcal{H} \setminus \mathcal{S}, I\rangle \models_{ht} F$ iff $\langle \mathcal{H}, I\rangle \models_{ht} F$.

About a model I of a set Γ of sentences over σ^I we say it is *stable* if, for every proper subset $\mathcal{H}$ of $I^{\downarrow}$, HT-interpretation $\langle \mathcal{H}, I\rangle$ does not satisfy Γ. In application to finite sets of formulas with a single sort, this definition of a stable model is equivalent to the definition in terms of the operator SM (Ferraris *et al.* 2011).

We say that I is *pointwise stable* if, for every element M of $I^{\downarrow}$, $\langle I^{\downarrow} \setminus \{M\}, I\rangle$ does not satisfy Γ.

https://doi.org/10.1017/S147106842300039X Published online by Cambridge University Press