Recursive Aggregates as Intensional Functions*

Jorge Fandinno[1], Zachary Hansen[1]

[1]University of Nebraska Omaha, NE, USA

Abstract

Recently, it was shown how to translate logic programs with aggregates into first-order formulas. In combination with the SM operator, this translation can be used to describe the `ASP-Core-2` semantics for this class of programs without relying on grounding. In this paper we prove that this same translation can be used to describe the semantics used by the answer set solver `clingo` when we interpret functions in an intensional way. We also state a similar conjecture regarding the semantics used by the answer set solver `dlv`. Recall that these solvers extend the `ASP-Core-2` semantics to programs that contain recursive aggregates. If this last conjecture is correct, the interpretation of aggregates used by `clingo` and `dlv` are exactly the same and the difference in their behaviour arises only from the different interpretations of implication (and negation).

Keywords

Answer Set Programming, Aggregates, First-order formalization

1. Introduction

Answer set programming (ASP) is a form of declarative logic programming well-suited to solving knowledge-intensive search and optimization problems [1]. Its success relies on the combination of a rich knowledge representation language with efficient solvers. Some of the most useful constructs of this language are *aggregates*: intuitively, these are functions that apply to sets. The semantics of aggregates have been extensively studied in the literature [2, 3, 4, 5, 6, 7, 8, 9, 10]. In most cases, papers rely on the idea of *grounding* — a process in which all variables are replaced by variable-free terms. Thus, first a program with variables is transformed into a propositional program, then the semantics of the propositional program are defined. This makes reasoning about programs with variables cumbersome. For instance, it prohibits using first-order theorem provers for verifying properties of programs as advocated by Fandinno et al. [11].

Though there are several approaches to describe the semantics of aggregates that bypass the need for grounding, most of these approaches only allow a restricted class of aggregates [12, 13] or use some extension of the logical language [14, 15, 16, 17]. Recently, Fandinno et al. [18] showed how to translate logic programs with aggregates into first-formulas, which, after the application of the SM operator [19] to the result, captures the `ASP-Core-2` semantics. Recall that the `ASP-Core-2` semantics do not allow recursion through aggregates. Despite the fact that most practical problems can be represented within the restrictions of the `ASP-Core-2` semantics, there are some notable execptions that are more naturally represented using recursive aggregates.

ASPOCP 2023: 16th Workshop on Answer Set Programming and Other Computing Paradigms

*This is original work.

jfandinno@unomaha.edu (J. Fandinno); zachhansen@unomaha.edu (Z. Hansen)

One of these examples is the *Company Control problem*, which consists of finding companies that control other companies by (directly or indirectly) owning a majority of its shares. This problem has been encoded in the literature using the following logic program [4, 7, 20, 21, 22]:

```
ctrStk(C1,C1,C2,P) :- ownsStk(C1,C2,P).                          (1)
ctrStk(C1,C2,C3,P) :- controls(C1,C2), ownsStk(C2,C3,P).         (2)
   controls(C1,C3) :- company(C1), company(C3),
                      #sum{P,C2 : ctrStk(C1,C2,C3,P)} > 50.      (3)
```

where `ownsStk(C1,C2,P)` means that company `C1` directly owns `P%` of the shares of company `C2`; `ctrStk(C1,C2,C3,P)` means that company `C1` controls `P%` of the shares of company `C3` through company `C2` that it controls; and `controls(C1,C3)` means that company `C1` controls company `C3`. Another area where allowing recursive aggregates is important is in the study of *strong equivalence* [23, 24]. Recall that the strong equivalence problem consists of determining whether two programs have the same behaviour in any context. Here, even if the program we are analyzing does not contain recursion, augmenting it by some context may introduce recursion.

In this paper, we show that the translation introduced by Fandinno et al. [18] can also be used for programs with recursive aggregates if we interpret functions in an intensional way [25, 26, 27, 28]. In particular, we focus on the Abstract Gringo [29] generalization of the semantics by Ferraris [6], which is used in the answer set solver `clingo`, and the semantics by Faber et al. [7], which is used in the answer set solver `dlv`. More precisely, we prove that the translation introduced by Fandinno et al. [18] coincides with the Abstract Gringo semantics when we interpret the function symbols representing sets according to the semantics for intensional functions by Bartholomew and Lee [28]. For the semantics used by `dlv`, we introduce a variation of these semantics for intensional functions based on the FLPT semantics [14] and state a similar conjecture, which is matter of ongoing study.

2. Preliminaries

Syntax of programs with aggregates. We assume a *(program) signature* with three countably infinite sets of symbols: *numerals*, *symbolic constants* and *program variables*. We also assume a 1-to-1 correspondence between numerals and integers; the numeral corresponding to an integer n is denoted by $\overline{n}$. *Program terms* are either numerals, symbolic constants, variables or either of the special symbols inf and sup. A program term (or any other expression) is *ground* if it contains no variables. We assume that a total order on ground terms is chosen such that

- inf is its least element and sup is its greatest element,
- for any integers m and n, $\overline{m} < \overline{n}$ iff $m < n$, and
- for any integer n and any symbolic constant c, $\overline{n} < c$.

An *atom* is an expression of the form $p(\mathbf{t})$, where p is a symbolic constant and $\mathbf{t}$ is a list of program terms. A *comparison* is an expression of the form $t \prec t'$, where t and t' are program terms and $\prec$ is one of the *comparison symbols*:

$$= \quad \neq \quad < \quad > \quad \leq \quad \geq \tag{4}$$

An *atomic formula* is either an atom or a comparison. A *basic literal* is an atomic formula possibly preceded by one or two occurrences of *not*. An *aggregate element* has the form

$$t_1, \ldots, t_k : l_1, \ldots, l_m \tag{5}$$

where each t_i ($1 \leq i \leq k$) is a program term and each l_i ($1 \leq i \leq m$) is a basic literal. An *aggregate atom* is of form

$$\#\mathrm{op}\{E\} \prec u \tag{6}$$

where op is an operation name, E is an aggregate element, $\prec$ is one of the comparison symbols in (1), and u is a program term, called *guard*. We consider operation names names `count` and `sum`. For example, expression

```
#sum{P,C2 : ctrStk(C1,C2,C3,P)} > 50
```

in the body of rule (3) is an aggregate atom. An *aggregate literal* is an aggregate atom possibly preceded by one or two occurrences of *not*. A *literal* is either a basic literal or an aggregate literal.

A *rule* is an expression of the form

$$Head \text{ :- } B_1, \ldots, B_n, \tag{7}$$

where

- $Head$ is either an atom or symbol $\bot$; we often omit symbol $\bot$ which results in an empty head;
- each B_i ($1 \leq i \leq n$) is a literal.

We call the symbol :- a *rule operator*. We call the left hand side of the rule operator the *head*, the right hand side of the rule operator the *body*. When the head of the rule is an atom we call the rule *normal*. A *program* is a finite set of rules.

Each operation name op is associated with a function $\widehat{\mathrm{op}}$ that maps every set of tuples of ground terms to a ground term. If the first member of a tuple $\mathbf{t}$ is a numeral $\overline{n}$ then we say that integer n is the weight of $\mathbf{t}$, otherwise the weight of $\mathbf{t}$ is 0. For any set Δ of tuples of ground terms,

- $\widehat{\mathtt{count}}(\Delta)$ is the numeral corresponding to the cardinality of Δ, if Δ is finite; and *sup* otherwise.
- $\widehat{\mathtt{sum}}(\Delta)$ is the numeral corresponding to the sum of the weights of all tuples in Δ, if Δ contains finitely many tuples with non-zero weights; and 0 otherwise.[1] If Δ is empty, then $\widehat{\mathtt{sum}}(\Delta) = 0$.

[1]The sum of a set of integers is not always defined. We could choose a special symbol to denote this case, we chose to use 0 following the description of abstract `gringo` [30].

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 $(F \rightarrow G) \wedge (G \rightarrow F)$.

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.

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.

3. Programs With Aggregates as Many-Sorted First-Order Sentences

In this section, we present translation τ^* that turns a program Π into a first-order sentence with equality over a signature $\sigma^*(\mathcal{P}, \mathcal{S})$ of *three sorts* where $\mathcal{P}$ and $\mathcal{S}$ respectively are sets of

predicate and *set symbols*. An *set symbol* is a pair $E/\mathbf{X}$, where E is an aggregate element and $\mathbf{X}$ is a list of variables occurring in E. In the sake of brevity, each set symbol $E/\mathbf{X}$ is assigned a short name $|E/\mathbf{X}|$. The first sort is called the *program sort* (denoted s_{prg}); all program terms are of this sort. The second sort is called the *tuple sort* (denoted s_{tuple}); it contains entities that are *tuples* of objects of the program sort. The second sort is called the *set sort* (denoted s_{set}); it contains entities that are *sets* of elements of the second sort, that is, tuples of objects of the program sort. Signature $\sigma^*(\mathcal{P}, \mathcal{S})$ contains:

1. all ground terms as object constants of the program sort;
2. all predicate symbols in $\mathcal{P}$ with all arguments of program sort;
3. the comparison symbols other than equality as binary predicate constants whose arguments are of the program sort;
4. predicate constant $\in/2$ with the first argument of the sort tuple and the second argument of the sort set;
5. function constant $tuple/k$ with arguments of the program sort and value of the tuple sort for each set symbol $E/\mathbf{X}$ in $\mathcal{S}$ with E of the form of (5);
6. unary function constants $count$ and sum whose argument is of the set sort and its value is of the program sort;
7. for each set symbol $E/\mathbf{X}$ in $\mathcal{S}$, a function constant $set_{|E/\mathbf{X}|}/n$ whose arguments are of the program sort and its value is of the set sort and where n is the number of variables in $\mathbf{X}$;

We use infix notation in constructing atoms that utilize predicate symbols $>, \geq, <, \leq, \neq$ and $\in$.

Intuitively, $tuple(t_1, \ldots, t_k)$ is a constructor for the k-tuple containing program terms $t_1, \ldots, t_k$. Furthermore, each function constant $set_{|E/\mathbf{X}|}/n$ maps an n-tuple of ground terms $\mathbf{x}$ to the set of tuples represented[2] by $E^{\mathbf{X}}_{\mathbf{x}}$. The result of $count$ is the cardinality of the set passed as an argument; the result of sum is the sum of the weights of all elements in the set passed as an argument. These claims are formalized below. Note also that in contrast to other work on the theory of stable models that do not consider aggregates (or intensional functions), expression $t_1 \neq t_2$ is not equivalent to $\neg(t_1 = t_2)$. This is necessary because according to the Abstract Gringo semantics inequality and the negation of equality are not equivalent when used in aggregate atoms. For instance, the program consisting of rule

```
p(a) :- not #count{X : p(X)} = 0.
```
(8)

has two answer sets according to the Abstract Gringo semantics: the empty set and $\{p(a)\}$. The program consisting of rule

```
p(a) :- #count{X : p(X)} ≠ 0.
```
(9)

has the empty set as its unique answer set. On the other hand, this replacement is correct under the `dlv` semantics and, in fact, both programs have only the empty set as their unique answer set.

[2]For a tuple $\mathbf{X}$ of distinct variables, a tuple $\mathbf{x}$ of ground terms of the same length as $\mathbf{X}$, and an expression α, by $\alpha^{\mathbf{X}}_{\mathbf{x}}$ we denote the expression obtained from α by substituting $\mathbf{x}$ for $\mathbf{X}$.

About a predicate symbol p/n, we say that is *occurs* in a program Π if there is an atom of the form $p(t_1, \ldots, t_n)$ in Π. For set symbols, we need to introduce first the concepts of a global variable and an set symbol. A variable is said to be *global* in a rule if

1. it occurs in any non-aggregate literal, or
2. it occurs in a guard of any aggregate literal.

We say that set symbol $E/\mathbf{X}$ occurs in rule R if this rule contains an aggregate literal with the aggregate element E and $\mathbf{X}$ is the list of all variables in E that are global in R. We say that $E/\mathbf{X}$ occurs in a program Π if $E/\mathbf{X}$ occurs in some rule of the program. For instance, in rule (3) the global variables are `C1` and `C3`. Set symbol $E_{ctr}/\mathbf{X}_{ctr}$ occurs in this rule where E_{ctr} stands for the aggregate element `P,C2 : ctrStk(C1,C2,C3,P)` and $\mathbf{X}_{ctr}$ is the list of variables `C1`, `C3`. In the following, we denote by $set_{ctr}/2$ the function symbol associated with this set symbol.

In the following, when discussing some program Π, we assume a signature $\sigma^*(\mathcal{P}, \mathcal{S})$ such that $\mathcal{P}$ is a set that contains all predicate symbols occurring in Π and $\mathcal{S}$ contains all set symbols occurring in Π. When no confusion arises we will simply write σ^* instead of $\sigma^*(\mathcal{P}, \mathcal{S})$. In the following, we also assume that $\mathcal{P}$ is the set of intensional symbols and that the set of intensional functions $\mathcal{F}$ is the set of all function symbols corresponding to the set symbols in $\mathcal{S}$.

Translation τ^* We now describe a translation τ^* that converts a program into a finite set of first-order sentences.

Given a list $\mathbf{Z}$ of global variables in some rule R, we define $\tau^*_{\mathbf{Z}}$ for all elements of R as follows:

1. for every atomic formula A occurring outside of an aggregate literal, its translation $\tau^*_{\mathbf{Z}}A$ is A itself; $\tau^*_{\mathbf{Z}}\bot$ is $\bot$;
2. for an aggregate atom A of form $\texttt{\#count}\{E\} \prec u$ or $\texttt{\#sum}\{E\} \prec u$, its translation $\tau^*_{\mathbf{Z}}$ is the atom

$$count(set_{|E/\mathbf{X}|}(\mathbf{X})) \prec u \text{ or } sum(set_{|E/\mathbf{X}|}(\mathbf{X})) \prec u$$

   respectively, where $\mathbf{X}$ is the list of variables in $\mathbf{Z}$ occurring in E;
3. for every (basic or aggregate) literal of the form $not\ A$ its translation $\tau^*_{\mathbf{Z}}(not\ A)$ is $\neg\tau^*_{\mathbf{Z}}A$; for every literal of the form $not\ not\ A$ its translation $\tau^*_{\mathbf{Z}}(not\ not\ A)$ is $\neg\neg\tau^*_{\mathbf{Z}}A$.

We now define the translation τ^* as follows:

4. for every rule R of form (4), its translation τ^*R is the universal closure of

$$\tau^*_{\mathbf{Z}}B_1 \wedge \cdots \wedge \tau^*_{\mathbf{Z}}B_n \rightarrow \tau^*_{\mathbf{Z}}\ Head,$$

   where $\mathbf{Z}$ is the list of the global variables of R.
5. for every program Π, its translation $\tau^*\Pi$ is the first-order theory containing τ^*R for each rule R in Π.

For instance, rule (3) is translated into

$$company(C_1) \wedge company(C_3) \wedge sum(set_{ctr}(C_1, C_3)) > 50 \rightarrow controls(C_1, C_3) \qquad (10)$$

where variables C_1 and C_3 are of the program sort.

Standard interpretations. A *standard interpretation* I is an interpretation of $\sigma^*(\mathcal{P}, \mathcal{F})$ that satisfies the following *conditions*:

1. the domain $|I|^{s_{prg}}$ is the set containing all ground terms of the program sort (or ground program terms, for short);
2. universe $|I|^{s_{tuple}}$ is the set of all tuples of form $\langle d_1, \ldots, d_k \rangle$ with $d_i \in |I|^{s_{prg}}$ for each set symbol $E/\mathbf{X}$ in $\mathcal{S}$ with E of the form of (5);
3. the elements of domain $|I|^{s_{set}}$ are sets of elements from $|I|^{s_{tuple}}$;
4. I interprets each ground program term as itself;
5. I interprets predicate symbols $>, \geq, <, \leq$ according to the total order chosen earlier;
6. I interprets each tuple term of form $tuple(t_1, \ldots, t_k)$ as the tuple $\langle t_1^I, \ldots, t_k^I \rangle$;
7. $\in^I$ is the set of pairs (t, s) such that tuple t belongs to set s;
8. for term t_{set} of sort s_{set}, $count(t_{set})^I$ is $\widehat{\texttt{count}}(t_{set}^I)$;
9. for term t_{set} of sort s_{set}, $sum(t_{set})^I$ is $\widehat{\texttt{sum}}(t_{set}^I)$;

To complete the first-order characterization of the Abstract Gringo semantics, we introduce a *set axiom* as follows. For each set symbol $E/\mathbf{X}$ of the form of (5), by $\mathrm{SCA}(E/\mathbf{X})$ we denote the first-order sentence

$$\forall \mathbf{X} T \big(T \in set_{|E/\mathbf{X}|}(\mathbf{X}) \leftrightarrow \exists \mathbf{Y} (T = tuple(t_1, \ldots, t_k) \wedge l_1 \wedge \cdots \wedge l_m) \big) \tag{11}$$

where $\mathbf{Y}$ is the list of all the variables occurring in E that are not in $\mathbf{X}$. We write SCA for the set containing $\mathrm{SCA}(E/\mathbf{X})$ for each set symbol $E/\mathbf{X}$ in the signature.

Proposition 1. *A standard interpretation I of σ^* satisfies (11) iff it satisfies the following condition:*

> *$set_{|E/\mathbf{X}|}(\mathbf{x})^I$ is the set of all tuples of form $\langle (t_1)^{\mathbf{XY}}_{\mathbf{xy}}, \ldots, (t_k)^{\mathbf{XY}}_{\mathbf{xy}} \rangle$ such that I satisfies $(l_1)^{\mathbf{XY}}_{\mathbf{xy}} \wedge \cdots \wedge (l_m)^{\mathbf{XY}}_{\mathbf{xy}}$*

where $\mathbf{x}$ and $\mathbf{y}$ are lists of ground program terms of the same length as $\mathbf{X}$ and $\mathbf{Y}$ respectively.

For instance, the program representing the Company Control problem has a unique set symbol that is associated with the function symbol $set_{ctr}/2$. For this function symbol, we have the first-order sentence

$$\forall C_1 C_3 T \big(T \in set_{ctr}(C_1, C_3) \leftrightarrow \exists P C_2 (T = tuple(P, C_2) \wedge ctrStk(C_1, C_2, C_3, P)) \big) \tag{12}$$

If I is a model of (12) such that $ctrStk^I$ is the set containing $(c1, c_2, c_3, 10)$ and $(c1, c_4, c_3, 20)$, by Proposition 1, it follows that $set_{ctr}(c_1, c_3)^I$ is the set containing tuples $\langle 10, c2 \rangle$ and $\langle 20, c4 \rangle$.

4. Abstract Gringo Semantics

In this section, we establish the correspondence between the semantics of programs with aggregates introduced in the previous section and the Abstract Gringo [29]. These semantics are stated in terms of infinitary formulas.

Background on Infinitary Formulas We recall some definitions of infinitary logic [31]. We denote a propositional signature (a set of propositional atoms) as σ. For every nonnegative integer r, *(infinitary propositional) formulas of rank* r are defined recursively:

- every ground atom in σ is a formula of rank 0,
- if Γ is a set of formulas, and r is the smallest nonnegative integer that is greater than the ranks of all elements of Γ, then $\Gamma^\wedge$ and $\Gamma^\vee$ are formulas of rank r,
- if F and G are formulas, and r is the smallest nonnegative integer that is greater than the ranks of F and G, then $F \to G$ is a formula of rank r.

We write $\{F, G\}^\wedge$ as $F \wedge G$, $\{F, G\}^\vee$ as $F \vee G$, and $\emptyset^\vee$ as $\bot$.

Subsets of a propositional signature σ will be called *interpretations*. The satisfaction relation between an interpretation $\mathcal{A}$ and an infinitary formula is defined recursively:

- for every ground atom A from σ, $\mathcal{A} \models A$ if A belongs to $\mathcal{A}$,
- $\mathcal{A} \models \Gamma^\wedge$ if for every formula F in Γ, $\mathcal{A} \models F$,
- $\mathcal{A} \models \Gamma^\vee$ if there is a formula F in Γ such that $\mathcal{A} \models F$,
- $\mathcal{A} \models F \to G$ if $\mathcal{A} \not\models F$ or $\mathcal{A} \models G$.

An interpretation satisfies a set Γ of formulas if it satisfies every formula in Γ. We say that a set $\mathcal{A}$ of atoms is a $\subseteq$-minimal model of an infinitary formula F, if $\mathcal{A} \models F$ and there is no $\mathcal{B}$ that satisfies both $\mathcal{B} \models F$ and $\mathcal{B} \subset \mathcal{A}$.

Stable Models The *FT-reduct* $F^\mathcal{A}$ of an infinitary formula F with respect to a set $\mathcal{A}$ of atoms is defined recursively. If $\mathcal{A} \not\models F$ then $F^\mathcal{A}$ is $\bot$; otherwise,

- for every ground atom A, $A^\mathcal{A}$ is A
- $(\Gamma^\wedge)^\mathcal{A} = \{G^\mathcal{A} \mid G \in \Gamma\}^\wedge$,
- $(\Gamma^\vee)^\mathcal{A} = \{G^\mathcal{A} \mid G \in \Gamma\}^\vee$,
- $(G \to H)^\mathcal{A}$ is $G^\mathcal{A} \to H^\mathcal{A}$.

We say that a set $\mathcal{A}$ of ground atoms is a *FT-stable model* of an infinitary formula F if it is a $\subseteq$-minimal model of $F^\mathcal{A}$. We say that a set $\mathcal{A}$ of ground atoms is a *gringo answer set* of a program Π if $\mathcal{A}$ a FT-stable model of $\tau\Pi$ where τ is the translation from logic programs to infinitary formulas defined by Gebser et al. [29].

5. First-order characterization of the Abstract Gringo Semantics

The first-order characterization of the Abstract Gringo Semantics proposed here relies on the stable model semantics with intensional functions [28]. Our presentation follows the definition in terms of of equilibrium logic [32] described by Bartholomew and Lee [33, Section 3.3].

Stable Model Semantics with Intensional Functions. Let I and H be two interpretations of a signature σ and $\mathcal{P}$ and $\mathcal{F}$ respectively be sets of predicate and function constants of σ. We write $H \leq^{\mathcal{PF}} I$ if

- H and I have the same universe for all sorts;
- $p^H \subseteq p^I$ for every predicate constant p in P and $p^H = p^I$ for every predicate constant p not in P;
- $f^H \neq f^I$ for every function constant p in F and $p^H = p^I$ for every function constant f not in F;

We write $H <^{\mathcal{PF}} I$ if $H \leq^{\mathcal{PF}} I$ and $H \neq I$.

Let $\mathcal{P}$ and $\mathcal{F}$ be sets of predicate and function constant of σ, that we consider *intensional*. An *ht-interpretation* of σ is a pair $\langle H, I\rangle$, where H and I are interpretation of σ such that $H \leq^{\mathcal{PF}} I$. (In terms of Kripke models with two sorts, I is the there-world, and H is the here-world of a non-total interpretation). The satisfaction relation $\models_{ht}$ between HT-interpretation $\langle H, I\rangle$ of σ and a sentence F over σ^I is defined recursively as follows:

- $\langle H, I\rangle \models_{ht} p(\mathbf{t})$, if $I \models p(\mathbf{t})$ and $H \models p(\mathbf{t})$;
- $\langle H, I\rangle \models_{ht} t_1 = t_2$ if $t_1^I = t_2^I$ and $t_1^H = t_2^H$;
- $\langle H, I\rangle \not\models_{ht} \bot$;
- $\langle H, I\rangle \models_{ht} F \wedge G$ if $\langle H, I\rangle \models_{ht} F$ and $\langle H, I\rangle \models_{ht} G$;
- $\langle H, I\rangle \models_{ht} F \vee G$ if $\langle H, I\rangle \models_{ht} F$ or $\langle H, I\rangle \models_{ht} G$;
- $\langle H, I\rangle \models_{ht} F \to G$ if
  (i) $I \models F \to G$, and
  (ii) $\langle H, I\rangle \not\models_{ht} F$ or $\langle H, I\rangle \models_{ht} G$,
- $\langle H, I\rangle \models_{ht} \forall X\, F(X)$ if $\langle H, I\rangle \models_{ht} F(d^*)$ for each $d \in |I|^s$, where s is the sort of X;
- $\langle H, I\rangle \models_{ht} \exists X\, F(X)$ if $\langle H, I\rangle \models_{ht} F(d^*)$ for some $d \in |I|^s$, where s is the sort of X.

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

An ht-interpretation $\langle H, I\rangle$ is said to be standard if both H and I are standard.

Proposition 2. *A standard ht-interpretation $\langle H, I\rangle$ of σ^* satisfies (11) iff I satisfies the condition stated in Proposition 1 and, in addition, $\langle H, I\rangle$ satisfies the following condition:*

> *$set_{|E/\mathbf{X}|}(\mathbf{x})^H$ is the set of all tuples of form $\langle (t_1)^{\mathbf{XY}}_{\mathbf{xy}}, \dots, (t_k)^{\mathbf{XY}}_{\mathbf{xy}}\rangle$ such that $\langle H, I\rangle$ satisfies $(l_1)^{\mathbf{XY}}_{\mathbf{xy}} \wedge \cdots \wedge (l_m)^{\mathbf{XY}}_{\mathbf{xy}}$*

where $\mathbf{x}$ and $\mathbf{y}$ are lists of ground program terms of the same length as $\mathbf{X}$ and $\mathbf{Y}$ respectively.

About a model I of a set Γ of sentences over σ^I we say it is *ht-stable* if every ht-interpretation $\langle H, I\rangle$ with $H <^{\mathcal{PF}} I$ does not satisfy Γ.

Now we define the first-order counterpart of gringo answer sets defined in the previous section. For an interpretation I of σ^*, by $Ans(I)$, we denote the set of ground atoms that are satisfied by I and whose predicate symbol is not a comparison. If I is a standard ht-stable model of SCA $\cup\, \tau^*\Pi$, we say that $Ans(I)$ is a *fo-gringo answer set* of Π.

Theorem 1. *The fo-gringo answer sets of any program coincide with its gringo answer sets.*

6. The dlv semantics

Similarly to the Abstract Gringo semantics, the dlv semantics can be stated in terms of a the same translation τ to infinitary formulas, but using a different reduct [34].

FLP-stable models are defined for sets of implications rather than arbitrary formulas. Let Γ be a set of infinitary formulas of the form $G \to H$, where H is a disjunction of propositional atoms from σ. The *FLP-reduct* $FLP(\Gamma, \mathcal{A})$ of Γ w.r.t. an interpretation $\mathcal{A}$ is the set of all formulas $G \to H$ from Γ such that $\mathcal{A}$ satisfies G. A set $\mathcal{A}$ of ground atoms is an *FLP-stable model* of Γ if it is a $\subseteq$-minimal model of $FLP(\Gamma, I)$. We say that a set $\mathcal{A}$ of ground atoms is a *dlv answer set* of a program Π if $\mathcal{A}$ a FLP-stable model of $\tau\Pi$.

First-order characterization of the dlv semantics. For the dlv semantics we introduce a new entailment relation $\models_{dlv}$, which is variation of the here-and-there entailment relation $\models_{ht}$ defined above. This relation is defined recursively, analogously to the relation $\models_{ht}$, with the only difference being the implication case:

- $\langle H, I\rangle \models_{dlv} F \to G$ if
  (i) $I \models F \to G$, and
  (ii) $H \not\models F$ or $I \not\models F$ or both $H \models G$ and $I \models G$;

If $\langle H, I\rangle \models_{ht} F$ holds, we say that $\langle H, I\rangle$ *dlv-staisfies* F and that $\langle H, I\rangle$ is an *dlv-model* of F.

Proposition 3. *A standard ht-interpretation $\langle H, I\rangle$ of σ^* dlv-staisfies (11) iff I satisfies the condition stated in Proposition 1 and, in addition, H satisfies the following condition:*

> *$set_{|E/\mathbf{X}|}(\mathbf{x})^H$ is the set of all tuples of form $\langle (t_1)^{\mathbf{XY}}_{\mathbf{xy}}, \ldots, (t_k)^{\mathbf{XY}}_{\mathbf{xy}}\rangle$ such that H satisfies $(l_1)^{\mathbf{XY}}_{\mathbf{xy}} \wedge \cdots \wedge (l_m)^{\mathbf{XY}}_{\mathbf{xy}}$*

where $\mathbf{x}$ and $\mathbf{y}$ are lists of ground program terms of the same length as $\mathbf{X}$ and $\mathbf{Y}$ respectively.

About a model I of a set Γ of sentences, we say it is *dlv-stable* if there is no ht-interpretation $\langle H, I\rangle$ with $H <^{\mathcal{PF}} I$ and $\langle H, I\rangle \models_{dlv} \Gamma$. If I is a dlv-stable model of SCA $\cup\, \tau^*\Pi$, we say that $Ans(I)$ is a *fo-dlv answer set* of Π.

The following conjecture about the relation between dlv answer sets and fo-dlv answer sets is a matter of ongoing work:

Conjecture 1. *The fo-dlv answer sets of any program coincide with its dlv answer sets.*

7. Conclusion

This work presents a characterization of recursive aggregates that does not rely on grounding. For this, we use the same translation from logic programs to first-order formulas described by Fandinno et al. [35], but we rely on semantics that treat function symbols corresponding to set symbols as intensional. Recall that these functions map ground terms to sets of tuples of ground terms - each of these tuples must satisfy the associated conditions present in the aggregate element. Interestingly, for the semantics of the solver clingo, this particular class of

function symbols behaves in a truly analogous way to intensional predicate symbols; namely, the constructed set in the here-world is a subset of the constructed set in the there-world, for any arguments to the function. This is not ordinarily the case for intensional functions nor does it seem to be the case for the semantics of the solver `dlv`.

Another interesting outcome of this work is Conjecture 1. The implication of this conjecture (if proven) is that the differences between recursive aggregates in `clingo` and `dlv` are not actually rooted in aggregates themselves. In both solvers, aggregates are the same intensional functions on sets of tuples, but which tuples belong to these sets varies between solvers due to differences in their treatment of implication and negation. The proof of this conjecture is a matter of ongoing work.

The results presented in this paper bring us closer to a long-term goal of using first-order reasoning tools to automatically verify properties of programs with aggregates. The next step is to prove Conjecture 1 and explore its implications. Subsequent future work will include extending results on strong equivalence to the class of programs studied here.

References

[1] V. Lifschitz, Answer Set Programming, Springer-Verlag, 2019.

[2] P. Simons, I. Niemelä, T. Soininen, Extending and implementing the stable model semantics, Artificial Intelligence 138 (2002) 181–234.

[3] A. Dovier, E. Pontelli, G. Rossi, Intensional sets in CLP, in: C. Palamidessi (Ed.), Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings, volume 2916 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 284–299.

[4] N. Pelov, M. Denecker, M. Bruynooghe, Well-founded and stable semantics of logic programs with aggregates, Theory and Practice of Logic Programming 7 (2007) 301–353.

[5] T. Son, E. Pontelli, A constructive semantic characterization of aggregates in answer set programming, Theory and Practice of Logic Programming 7 (2007) 355–375.

[6] P. Ferraris, Logic programs with propositional connectives and aggregates, ACM Transactions on Computational Logic 12 (2011) 25:1–25:40.

[7] W. Faber, G. Pfeifer, N. Leone, Semantics and complexity of recursive aggregates in answer set programming, Artificial Intelligence 175 (2011) 278–298.

[8] M. Gelfond, Y. Zhang, Vicious circle principle and logic programs with aggregates, Theory and Practice of Logic Programming 14 (2014) 587–601.

[9] M. Gelfond, Y. Zhang, Vicious circle principle, aggregates, and formation of sets in ASP based languages, Artificial Intelligence 275 (2019) 28–77.

[10] P. Cabalar, J. Fandinno, T. Schaub, S. Schellhorn, Gelfond-zhang aggregates as propositional formulas, Artificial Intelligence 274 (2019) 26–43.

[11] J. Fandinno, V. Lifschitz, P. Lühne, T. Schaub, Verifying tight logic programs with anthem and vampire, Theory and Practice of Logic Programming 20 (2020) 735–750.

[12] J. Lee, V. Lifschitz, R. Palla, A reductive semantics for counting and choice in answer set programming, in: D. Fox, C. Gomes (Eds.), Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08), AAAI Press, 2008, pp. 472–479.

[13] V. Lifschitz, Strong equivalence of logic programs with counting, Theory and Practice of Logic Programming 22 (2022) 573–588.
[14] M. Bartholomew, J. Lee, Y. Meng, First-order extension of the flp stable model semantics via modified circumscription., in: T. Walsh (Ed.), Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI'11), IJCAI/AAAI Press, 2011, pp. 724–730.
[15] J. Lee, Y. Meng, Stable models of formulas with generalized quantifiers (preliminary report), in: A. Dovier, V. Santos Costa (Eds.), Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12), volume 17, Leibniz International Proceedings in Informatics (LIPIcs), 2012, pp. 61–71.
[16] V. Asuncion, Y. Chen, Y. Zhang, Y. Zhou, Ordered completion for logic programs with aggregates, Artificial Intelligence 224 (2015) 72–102.
[17] P. Cabalar, J. Fandinno, L. Fariñas del Cerro, D. Pearce, Functional ASP with intensional sets: Application to Gelfond-Zhang aggregates, Theory and Practice of Logic Programming 18 (2018) 390–405.
[18] J. Fandinno, Z. Hansen, Y. Lierler, Axiomatization of aggregates in answer set programming, in: Proceedings of the Thirty-sixth National Conference on Artificial Intelligence (AAAI'22), AAAI Press, 2022, pp. 5634–5641.
[19] P. Ferraris, J. Lee, V. Lifschitz, Stable models and circumscription, Artificial Intelligence 175 (2011) 236–263.
[20] I. Mumick, H. Pirahesh, R. Ramakrishnan, The magic of duplicates and aggregates, in: D. McLeod, R. Sacks-Davis, H. Schek (Eds.), Proceedings of the Sixteenth International Conference on Very Large Data Bases (VLDB'90), Morgan Kaufmann Publishers, 1990, pp. 264–277.
[21] D. Kemp, P. Stuckey, Semantics of logic programs with aggregates, in: V. Saraswat, K. Ueda (Eds.), Proceedings of the 1991 International Symposium on Logic Programming (ISLP'91), MIT Press, 1991, pp. 387–401.
[22] K. Ross, Y. Sagiv, Monotonic aggregation in deductive databases, Journal of Computer and System Sciences 54 (1997) 79–97.
[23] V. Lifschitz, D. Pearce, A. Valverde, Strongly equivalent logic programs, ACM Transactions on Computational Logic 2 (2001) 526–541.
[24] V. Lifschitz, D. Pearce, A. Valverde, A characterization of strong equivalence for logic programs with variables, in: C. Baral, G. Brewka, J. Schlipf (Eds.), Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), volume 4483 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2007, pp. 188–200.
[25] F. Lin, Y. Wang, Answer set programming with functions, in: G. Brewka, J. Lang (Eds.), Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08), AAAI Press, 2008, pp. 454–465.
[26] P. Cabalar, Functional answer set programming, Theory and Practice of Logic Programming 11 (2011) 203–233.
[27] M. Balduccini, ASP with non-herbrand partial functions: A language and system for practical use, Theory and Practice of Logic Programming 13 (2013) 547–561.
[28] M. Bartholomew, J. Lee, First-order stable model semantics with intensional functions,

Artificial Intelligence 273 (2019) 56–93.
[29] M. Gebser, A. Harrison, R. Kaminski, V. Lifschitz, T. Schaub, Abstract Gringo, Theory and Practice of Logic Programming 15 (2015) 449–463. doi:10.1017/S1471068415000150.
[30] M. Gebser, A. Harrison, R. Kaminski, V. Lifschitz, T. Schaub, Abstract gringo, Theory and Practice of Logic Programming 15 (2015) 449–463. doi:10.1017/s1471068415000150.
[31] M. Truszczyński, Connecting first-order ASP and the logic FO(ID) through reducts, in: E. Erdem, J. Lee, Y. Lierler, D. Pearce (Eds.), Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz, volume 7265 of *Lecture Notes in Computer Science*, Springer-Verlag, 2012, pp. 543–559.
[32] D. Pearce, A. Valverde, Quantified equilibrium logic and foundations for answer set programs, in: M. Garcia de la Banda, E. Pontelli (Eds.), Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08), volume 5366 of *Lecture Notes in Computer Science*, Springer-Verlag, 2008, pp. 546–560. doi:10.1007/978-3-540-89982-2_46.
[33] M. Bartholomew, J. Lee, On the stable model semantics for intensional functions, Theory and Practice of Logic Programming 13 (2013) 863–876.
[34] A. Harrison, V. Lifschitz, Relating two dialects of answer set programming, Theory and Practice of Logic Programming 19 (2019) 1006–1020.
[35] J. Fandinno, Z. Hansen, Y. Lierler, Axiomatization of aggregates in answer set programming, Proceedings of the AAAI Conference on Artificial Intelligence 36 (2022) 5634–5641. doi:10.1609/aaai.v36i5.20504.

A. Proof of Results

Lemma 1. *Let E be an aggregate element of the form (5) with free variables $\mathbf{X}$ and bound variables $\mathbf{Y}$ and let I be a standard interpretation. Let $\mathbf{x}$ be a list of ground terms of sort s_{prg} of the same length as $\mathbf{X}$, d_{tuple} a domain element of sort tuple, $l'_i = (l_i)^{\mathbf{X}}_{\mathbf{x}}$ and $t'_i = (t_i)^{\mathbf{X}}_{\mathbf{x}}$. Then, I satisfies*

$$d^*_{tuple} \in set_{|E|}(\mathbf{x}) \leftrightarrow \exists \mathbf{Y}\ \left(d^*_{tuple} = tuple(t'_1, \dots, t'_m) \wedge l'_1 \wedge \cdots \wedge l'_n\right) \tag{13}$$

iff the first of the following three conditions is equivalent to the conjunction of the other two:

1. *d_{tuple} belongs to $set_{|E|}(\mathbf{x})^I$,*
2. *there is a list $\mathbf{c}$ of domain elements of sort s_{prg} of the same length as $\mathbf{Y}$ such that $d_{tuple} = \langle (t''_1)^I, \dots, (t''_m)\rangle^I$ and I satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $t''_i = (t'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $l''_i = (l'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $\mathbf{y} = \mathbf{c}^*$.*

Proof. I satisfies (13) iff condition 1 is equivalent to:

$$I \text{ satisfies } \exists \mathbf{Y}\ (T = tuple(t'_1, \dots, t'_m) \wedge l'_1 \wedge \cdots \wedge l'_n)$$

Furthemore, the latter holds iff there is a list $\mathbf{c}$ of domain elements of sort s_{prg} such that

$$d_{tuple} \ = \ tuple(t''_1, \dots, t''_m)^I \ = \ tuple(t''_1, \dots, t''_m)^H \ = \ \langle (t''_1)^I, \dots, (t''_m)^I\rangle$$

and I satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $\mathbf{y} = \mathbf{c}^*$ iff condition 2 holds. □

Lemma 2. *Let E be an aggregate element of the form (5) with free variables $\mathbf{X}$ and bound variables $\mathbf{Y}$ and let I be a standard interpretation. Let $\mathbf{x}$ be a list of ground terms of sort s_{prg} of the same length as $\mathbf{X}$, $l'_i = (l_i)^{\mathbf{X}}_{\mathbf{x}}$ and $t'_i = (t_i)^{\mathbf{X}}_{\mathbf{x}}$. Then, I satisfies*

$$\forall T\big(T \in set_{|E|}(\mathbf{x}) \leftrightarrow \exists \mathbf{Y}\ (T = tuple(t'_1, \ldots, t'_m) \wedge l'_1 \wedge \cdots \wedge l'_n)\big) \tag{14}$$

iff $set_{|E|}(\mathbf{x})^I$ is the set of all tuples of the form $\langle (t''_1)^I, \ldots, (t''_m)^I \rangle$ s.t. I satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $t''_i = (t'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $l''_i = (l'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $\mathbf{y}$ a list of ground terms of sort s_{prg} of the same length as $\mathbf{Y}$.

Proof. Left-to-right. Assume that I satisfies (14) and pick any domain element d_{tuple} of sort s_{tuple}. Then, I satisfies (13) and, by Lemma 1, d_{tuple} belongs to $set_{|E|}(\mathbf{x})^I$ iff there is a list $\mathbf{c}$ of domain elements of sort s_{prg} of the same length as $\mathbf{Y}$ such that $d_{tuple} = \langle (t''_1)^I, \ldots, (t''_m) \rangle^I$ and I satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $t''_i = (t'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $l''_i = (l'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $\mathbf{y} = \mathbf{c}^*$. Then, the rigth-hand side of the iff holds.

Right-to-left. Assume that $set_{|E|}(\mathbf{x})^I$ is the set of all tuples of the form $\langle (t''_1)^I, \ldots, (t''_m)^I \rangle$ such that I satisfies $l''_1 \wedge \cdots \wedge l''_n$ for some list $\mathbf{y}$ of ground terms of sort s_{prg} of the same length as $\mathbf{Y}$. Pick any domain element d_{tuple} of sort s_{tuple}. By Lemma 1, I satisfies (13) and, thus, I satisfies (14). □

Proof of Proposition 1. Pick any list $\mathbf{c}$ of domain elements of sort s_{prg} and let $\mathbf{x} = \mathbf{c}^*$. Then, the result follows directly by Lemma 2. □

Lemma 3. $\langle H, I \rangle \models_{ht} F \leftrightarrow G$ *iff*

- $I \models F \leftrightarrow G$, *and*
- $\langle H, I \rangle \models_{ht} F$ *iff* $\langle H, I \rangle \models_{ht} G$.

Proof. $\langle H, I \rangle \models_{ht} F \leftrightarrow G$
iff $\langle H, I \rangle \models_{ht} F \rightarrow G$ and $\langle H, I \rangle \models_{ht} G \rightarrow F$
iff both

- $I \models F \rightarrow G$ and either $\langle H, I \rangle \not\models_{ht} F$ or $\langle H, I \rangle \models_{ht} G$, and
- $I \models G \rightarrow F$ and either $\langle H, I \rangle \not\models_{ht} G$ or $\langle H, I \rangle \models_{ht} F$

iff

- $I \models F \rightarrow G$ and $I \models G \rightarrow F$, and
- either $\langle H, I \rangle \not\models_{ht} F$ or $\langle H, I \rangle \models_{ht} G$, and
- either $\langle H, I \rangle \not\models_{ht} G$ or $\langle H, I \rangle \models_{ht} F$

iff

- $I \models F \leftrightarrow G$, and
- $\langle H, I \rangle \models_{ht} F$ iff $\langle H, I \rangle \models_{ht} G$. □

Lemma 4. *If $\langle H, I \rangle$ is a standard ht-interpretation, then $set_{|E/\mathbf{X}|}(\mathbf{x})^H$ is a subset of $set_{E/\mathbf{X}}(\mathbf{x})^I$ for every set symbol $E/\mathbf{X}$ and list $\mathbf{x}$ of ground terms of the same length as $\mathbf{X}$.*

Proof. Pick any element d_{tuple} of $set_{|E/\mathbf{X}|}(\mathbf{x})^H$. Since $\langle H, I\rangle$ is a standard ht-interpretation, it follows that H is a standard interpreation and, thus, that the pair $(d_{tuple}, set_{|E/\mathbf{X}|}(\mathbf{x})^H)$ belongs to $\in^H$. Hence, $(d_{tuple}, set_{|E/\mathbf{X}|}(\mathbf{x})^H)$ also belongs to $\in^I$ and, thus, d_{tuple} of $set_{|E/\mathbf{X}|}(\mathbf{x})^I$. □

Lemma 5. *Let E be an aggregate element of the form (5) with free variables $\mathbf{X}$ and bound variables $\mathbf{Y}$ and let $\langle H, I\rangle$ be a standard ht-interpretation. Let $\mathbf{x}$ be a list of ground terms of sort s_{prg} of the same length as $\mathbf{X}$, d_{tuple} a domain element of sort tuple, $l'_i = (l_i)^{\mathbf{X}}_{\mathbf{x}}$ and $t'_i = (t_i)^{\mathbf{X}}_{\mathbf{x}}$. Then, $\langle H, I\rangle$ satisfies (13) iff the the following two conditions are equivalent:*

1. *d_{tuple} belongs to $set_{|E|}(\mathbf{x})^I$,*
2. *there is a list $\mathbf{c}$ of domain elements of sort s_{prg} of the same length as $\mathbf{Y}$ such that $d_{tuple} = \langle (t''_1)^I, \ldots, (t''_m)\rangle^I$ and I satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $t''_i = (t'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $l''_i = (l'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $\mathbf{y} = \mathbf{c}^*$,*

and the the following two conditions are aslo equivalent:

3. *d_{tuple} belongs to $set_{|E|}(\mathbf{x})^H$,*
4. *there is a list $\mathbf{c}$ of domain elements of sort s_{prg} of the same length as $\mathbf{Y}$ such that $d_{tuple} = \langle (t''_1)^I, \ldots, (t''_m)\rangle^H$ and $\langle H, I\rangle$ satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $t''_i = (t'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $l''_i = (l'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $\mathbf{y} = \mathbf{c}^*$.*

Proof. From Lemma 3, $\langle H, I\rangle$ satisfies (13) iff I satisfies (13) and the following two conditions are equivalent:

5. $\langle H, I\rangle$ satisfies $d^*_{tuple} \in set_{|E|}(\mathbf{x})$;
6. $\langle H, I\rangle$ satisfies $\exists \mathbf{Y}\ (T = tuple(t'_1, \ldots, t'_m) \wedge l'_1 \wedge \cdots \wedge l'_n)$

Furthemore, by Lemma 1, I satisfies (13) iff Conditions 1 and 2 are equivalent. Hence, $\langle H, I\rangle$ satisfies (13) iff Conditions 1 and 2 are equivalent, and Conditions 5 and 6 are equivalent.

On the one hand, condition 6 is equivalent to conjuntion of conditions 1 and 3. On the other hand, condition 6 holds iff there is a list $\mathbf{c}$ of domain elements of sort s_{prg} such that

$$d_{tuple} \;=\; tuple(t''_1, \ldots, t''_m)^I \;=\; tuple(t''_1, \ldots, t''_m)^H \;=\; \langle (t''_1)^I, \ldots, (t''_m)^I\rangle$$

and $\langle H, I\rangle$ satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $\mathbf{y} = \mathbf{c}^*$ iff condition 4 holds. Therefore, $\langle H, I\rangle$ satisfies (13) iff

- Conditions 1 and 2 are equivalent, and
- Condition 4 is equivalent to the conjunction of conditions 1 and 3.

Finally, note that by Lemma 4 condition 3 implies 1 and the result holds. □

Lemma 6. *Let E be an aggregate element of the form (5) with free variables $\mathbf{X}$ and bound variables $\mathbf{Y}$ and let I be a standard interpretation. Let $\mathbf{x}$ be a list of ground terms of sort s_{prg} of the same length as $\mathbf{X}$, $l'_i = (l_i)^{\mathbf{X}}_{\mathbf{x}}$ and $t'_i = (t_i)^{\mathbf{X}}_{\mathbf{x}}$. Then, $\langle H, I\rangle$ satisfies (14) iff I satisfies the condition stated in Lemma 2 and, in addition, $set_{|E|}(\mathbf{x})^H$ is the set of all tuples of the form $\langle (t''_1)^I, \ldots, (t''_m)^I\rangle$ s.t. I satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $t''_i = (t'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $l''_i = (l'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $\mathbf{y}$ a list of ground terms of sort s_{prg} of the same length as $\mathbf{Y}$.*

Proof. Left-to-right. Assume that $\langle H, I\rangle$ satisfies (14) and pick any domain element d_{tuple} of sort s_{tuple}. Then, $\langle H, I\rangle$ satisfies (13) and the result follows immediately by Lemma 5.

Right-to-left. Pick any domain element d_{tuple} of sort tuple. The conditions of Lemma 5 are satisfied and the result follows by this lemma. □

Proof of Proposition 2. Pick any list $\mathbf{c}$ of domain elements of sort s_{prg} and let $\mathbf{x} = \mathbf{c}^*$. Then, the result follows directly by Lemma 2. □

A.1. Proof of Theorem 1

First-order interpretations and infinitary formulas. We can extend the satisfaction relation for ht-interpretations can be extended to infinitary formulas by adding the following two conditions to the definition for first-order formulas:

- $\langle H, I\rangle \models_{ht} \Gamma^{\wedge}$ if for every formula F in Γ, $\langle H, I\rangle \models_{ht} F$,
- $\langle H, I\rangle \models_{ht} \Gamma^{\vee}$ if there is a formula F in Γ such that $\langle H, I\rangle \models_{ht} F$,

We write $I \models F$ if $\langle I, I\rangle \models_{ht} F$.

In the following, if I is an interpretation, then $\mathcal{I}$ denotes the set of ground atomic formulas satisfied by I.

Proposition 4. *Let σ^p be a propositional signature consisting of all the ground atomic formulas of first-order signature σ. Let F be a an infinitary formula of σ^p. Then,*

$$\begin{array}{ccc} I \models F & \textit{iff} & \mathcal{I} \models F \\ \langle H, I\rangle \models_{ht} F & \textit{iff} & \mathcal{H} \models F^{\mathcal{I}} \end{array}$$

Proof. We proceed by induction on the rank r of F. For a formula F of rank $r+1$, assume that, for all formulas G of lesser rank than F occurring in F, $\langle H, I\rangle \models_{ht} G$ iff $\mathcal{H} \models G^{\mathcal{I}}$.

Base Case: $r = 0$, F is a ground atomic formula. Then, $\langle H, I\rangle \models_{ht} F$
iff $H \models F$ and $I \models F$
iff $H \models F$ and $F^{\mathcal{I}} = F$
iff $\mathcal{H} \models F^{\mathcal{I}}$

Induction Step:

Case 1: Formula F of rank $r+1$ has form $\Gamma^{\wedge}$. Then, $\langle H, I\rangle \models_{ht} F$
iff $\langle H, I\rangle \models_{ht} G$ for every formula G in Γ (by definition)
iff $\mathcal{H} \models G^{\mathcal{I}}$ for every formula G in Γ (by induction hypothesis)
iff $\mathcal{H} \models \{G^{\mathcal{I}} \mid G \in \Gamma\}^{\wedge}$
iff $\mathcal{H} \models F^{\mathcal{I}}$

Case 2: Formula F of rank $r+1$ has form $\Gamma^{\vee}$. Then, $\langle H, I\rangle \models_{ht} F$
iff $\langle H, I\rangle \models_{ht} G$ for some formula G in Γ (by definition)
iff $\mathcal{H} \models G^{\mathcal{I}}$ for this certain formula G in Γ (by induction hypothesis)
iff $\mathcal{H} \models \{G^{\mathcal{I}} \mid G \in \Gamma\}^{\vee}$
iff $\mathcal{H} \models F^{\mathcal{I}}$

Case 3: Formula F of rank $r+1$ has form $G_1 \to G_2$. Then, $\langle H, I\rangle \models_{ht} F$
iff $I \models G_1 \to G_2$ and $\langle H, I\rangle \not\models_{ht} G_1$ or $\langle H, I\rangle \models_{ht} G_2$ (by definition)
iff $\langle I, I\rangle \models_{ht} G_1 \to G_2$ and $\langle H, I\rangle \not\models_{ht} G_1$ or $\langle H, I\rangle \models_{ht} G_2$ (by definition)
iff $\mathcal{I} \models G_1^{\mathcal{I}} \to G_2^{\mathcal{I}}$ and $\mathcal{H} \not\models G_1^{\mathcal{I}}$ or $\mathcal{H} \models G_2^{\mathcal{I}}$ (by induction hypothesis)
iff $\mathcal{I} \models G_1 \to G_2$ and $\mathcal{H} \not\models G_1^{\mathcal{I}}$ or $\mathcal{H} \models G_2^{\mathcal{I}}$
iff $\mathcal{I} \models G_1 \to G_2$ and $\mathcal{H} \models G_1^{\mathcal{I}} \to G_2^{\mathcal{I}}$
iff $F^{\mathcal{I}} = G_1^{\mathcal{I}} \to G_2^{\mathcal{I}}$ and $\mathcal{H} \models G_1^{\mathcal{I}} \to G_2^{\mathcal{I}}$
iff $\mathcal{H} \models F^{\mathcal{I}}$ □

An interpretation that satisfies the condition of Proposition 1 for every set symbol $E/\mathbf{X}$ in $\mathcal{S}$ is called an agg-interpretation. Similarly, a ht-interpretation that satisfies the condition of Proposition 2 for every set symbol $E/\mathbf{X}$ in $\mathcal{S}$ is called an agg-ht-interpretation. A model that is an agg-interpretation is called an agg-model and a ht-model that is an agg-ht-interpretation is called an agg-ht-model.

Lemma 7. *Let $\langle H, I\rangle$ be an agg-ht-interpretation. Then, $H <^{\mathcal{PF}} I$ iff $H <^{\mathcal{P}\emptyset} I$.*

Proof. Right-to-left. $H <^{\mathcal{P}\emptyset} I$ means that there is a predicate symbol p such that $p^H \subset p^I$ and, thus, $H <^{\mathcal{PF}} I$ also holds. *Lert-to-right.* $H <^{\mathcal{PF}} I$ means that on of the following holds

- $p^H \subset p^I$ for some intensional predicate symbol p; or
- $f^H \neq f^I$ for some intensional function symbol f.

The first immediately implies that $H <^{\mathcal{P}\emptyset} I$ also holds. For the latter, f must be of the form $set_{|E/\mathbf{X}|}$ for some aggregate element E. Therefore, the set of the set of all tuples of form $\langle (t_1)^{\mathbf{XY}}_{\mathbf{xy}}, \ldots, (t_k)^{\mathbf{XY}}_{\mathbf{xy}}\rangle$ such that I satisfies $(l_1)^{\mathbf{XY}}_{\mathbf{xy}} \wedge \cdots \wedge (l_m)^{\mathbf{XY}}_{\mathbf{xy}}$ and the set of all tuples of form $\langle (t_1)^{\mathbf{XY}}_{\mathbf{xy}}, \ldots, (t_k)^{\mathbf{XY}}_{\mathbf{xy}}\rangle$ such that $\langle H, I\rangle$ satisfies $(l_1)^{\mathbf{XY}}_{\mathbf{xy}} \wedge \cdots \wedge (l_m)^{\mathbf{XY}}_{\mathbf{xy}}$ must be different. This means that $p^H \neq p^I$ for some predicate symbols p and, thus, $p^H \subset p^I$ and $H <^{\mathcal{P}\emptyset} I$ follow. □

We say that an agg-model I of an infinitary formula F is an *FT-agg-stable model* of F if there is no agg-ht-interpretation $\langle H, I\rangle$ with $H <^{\mathcal{PF}} I$ such that $\langle H, I\rangle$ satisfies F.

In the following two Lemmas, I is an agg-interpretationand $\mathcal{I}$ is the set of ground atomic formulas satisfied by I.

Lemma 8. *If $\mathcal{I}$ is a FT-stable model of F, then I is a FT-agg-stable model of F.*

Proof. Since I is a FT-stable model of F, we get that I satisfies F and, thus, I is an agg-model I of F. Suppose, for the sake of contradiction, that there is an agg-ht-interpretation $\langle H, I\rangle$ with $H <^{\mathcal{PF}} I$ such that $\langle H, I\rangle$ satisfies F. Then, by Proposition 4, it follows that $\mathcal{H} \models F^{\mathcal{I}}$. Furthermore, by Proposition 7, it follows that $H <^{\mathcal{P}\emptyset} I$ and, thus, $\mathcal{H} \subset \mathcal{I}$. This is a contradiction with the assumption that $\mathcal{I}$ is a FT-stable model of F. □

Lemma 9. *Let F be a an infinitary formula that does not contain function symbols corresponding to set symbols. Then, $\mathcal{I}$ is a FT-stable model of F iff I is FT-agg-stable model of F*

Proof. The left-to-right direction follows by Lemma 8. For the right-to-left direction, assume that I is an FT-agg-stable model of F. Then, I satisfies F and, by Proposition 4, it follows that $\mathcal{I}$ is a model of F. Suppose, for the sake of contradiction, that $\mathcal{I}$ is a FT-stable model of F. Then, there is some model $\mathcal{J}$ of $F^{\mathcal{I}}$ such that $\mathcal{J} \subset \mathcal{I}$. Let H be an agg-interpretation such that

- H has the same domain as I,
- for every ground atomic formula $p(\mathbf{t})$ that does not function symbols corresponding to set symbols, $p(\mathbf{t})^H$ is true iff $p(\mathbf{t})$ belongs to $\mathcal{J}$.

Then, $H <^{\mathcal{P}\emptyset} I$ and, from Proposition 7, we get that $H <^{\mathcal{PF}} I$. Since I is an FT-agg-stable model of F, it follows that $\langle H, I\rangle \not\models_{ht} F$. From Proposition 4, this implies that $\mathcal{H} \not\models F^{\mathcal{I}}$ where $\mathcal{H}$ is the set of ground atomic formulas satisfied by H. However, $\mathcal{J}$ and $\mathcal{H}$ agree on all atoms occurring in F, so this is a contradiction with the fact that $\mathcal{J}$ is a model of $F^{\mathcal{I}}$. □

Infinitary grounding The *grounding of a first order sentence F with respect to an interpretation I and sets $\mathcal{P}$ and $\mathcal{F}$ of intensional predicate and function symbols* is defined as follows:

- $gr_I^{\mathcal{PF}}(\bot) = \bot$;
- $gr_I^{\mathcal{PF}}(p(\mathbf{t})) = p((\mathbf{t}^I)^*)$ if $p(\mathbf{t})$ contains intensional symbols;
- $gr_I^{\mathcal{PF}}(p(\mathbf{t})) = \top$ if $p(\mathbf{t})$ does not contain intensional symbols and $I \models p(\mathbf{t})$; and $gr_I^{\mathcal{PF}}(p(\mathbf{t})) = \bot$ otherwise;
- $gr_I^{\mathcal{PF}}(t_1 = t_2) = (t_1 = t_2)$ if t_1 or t_2 contain intensional symbols;
- $gr_I^{\mathcal{PF}}(t_1 = t_2) = \top$ if t_1 and t_2 do not contain intensional symbols and $t_1^I = t_2^I$ and $\bot$ otherwise;
- $gr_I^{\mathcal{PF}}(F \otimes G) = gr_I^{\mathcal{PF}}(F) \otimes gr_I^{\mathcal{PF}}(G)$ if $\otimes$ is $\wedge$, $\vee$, or $\rightarrow$;
- $gr_I^{\mathcal{PF}}(\exists X\, F(X)) = \{gr_I^{\mathcal{PF}}(F(u)) \mid u \in |I|^s\}^{\vee}$ if X is a variable of sort s;
- $gr_I^{\mathcal{PF}}(\forall X\, F(X)) = \{gr_I^{\mathcal{PF}}(F(u)) \mid u \in |I|^s\}^{\wedge}$ if X is a variable of sort s;

For a first order theory Γ, we define $gr_I^{\mathcal{PF}}(\Gamma) = \{gr_I^{\mathcal{PF}}(F) \mid F \in \Gamma\}^{\wedge}$.

In the following, we write $gr_I(F)$ instead of $gr_I^{\mathcal{PF}}(F)$ when clear by the context.

Lemma 10. *An interpretation I satisfies a sentence F over σ^I iff I satisfies $gr_I(F)$.*

Proof. By induction on F.

Case 1: F is an atomic sentence. Then, $gr_I(F) = F$ and the result is trivial.

Case 2: F is $\forall X G(X)$ with X a variable of sort s. Then, $gr_I(F) = \{gr_I(G(d^*)) \mid d^* \in |I|^s\}^{\wedge}$ and

$\langle H, I\rangle \models_{ht} F$
iff $\langle H, I\rangle \models_{ht} G(d^*)$ for each $d \in |I|^s$
iff $\langle H, I\rangle \models_{ht} gr_I(G(d^*))$ for each $d \in |I|^s$ (induction)
iff $\langle H, I\rangle \models_{ht} gr_I(F)$.

The case where F is $\exists X G(X)$ is analogous to Case 2. The remaining cases where F is $G_1 \wedge G_2$, $G_1 \vee G_2$ or $F_1 \rightarrow F_2$ follow immediately by induction. □

Lemma 11. *An ht-interpretation $\langle H, I\rangle$ satisfies a sentence F over σ^I iff $\langle H, I\rangle$ satisfies $gr_I(F)$.*

Proof. By induction on F similar to Lemma 10. We show here the only case where F is $G_1 \to G_2$.
Then, $gr_I(F)$ is $gr_I^{\mathcal{PF}}(G_1) \to gr_I(G_2)$ and
$\langle H, I\rangle \models_{ht} F$
iff $I \models F$ and either $\langle H, I\rangle \not\models_{ht} G_1$ or $\langle H, I\rangle \models_{ht} G_2$
iff $I \models gr_I(F)$ (Lemma 10) and either $\langle H, I\rangle \not\models_{ht} gr_I^{\mathcal{PF}}(G_1)$
or $\langle H, I\rangle \models_{ht} gr_I(G_2)$ (induction)
iff $\langle H, I\rangle \models_{ht} gr_I(F)$. □

We say that an agg-model I of set Γ of a first-order sentences is an *agg-stable model* of Γ if there is no agg-ht-interpretation $\langle H, I\rangle$ with $H <^{\mathcal{PF}} I$ such that $\langle H, I\rangle$ satisfies $Gamma$.

Lemma 12. *Let Γ be a set of first-order sentences. Then, I is a agg-stable model of Γ iff I is an FT-agg-stable model of $gr_I(\Gamma)$.*

Proof. First, note that by Lemma 10, it follows that I is agg-model of Γ iff I is an agg-interpretation and $I \models \Gamma$ iff I is agg-modelof $gr_I(\Gamma)$. Then, I is a agg-stable model of Γ if there is no $\langle H, I\rangle$ with $H <^{\mathcal{PF}} I$ that satisfies Γ iff there is no $\langle H, I\rangle$ with $H <^{\mathcal{PF}} I$ that satisfies $gr_I(\Gamma)$ (Lemma 11) iff I is a FT-agg-stable model of $gr_I(\Gamma)$. □

Lemma 13. *Let I be an agg-interpretation and* op *be an operation name.*
Then, I satisfies $\mathsf{op}(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u$ *iff I satisfies*

$$\bigwedge_{\Delta\in\chi}\left(\bigwedge_{\mathbf{y}\in\Delta}\mathbf{l}_{\mathbf{xy}}^{\mathbf{XY}} \to \bigvee_{\mathbf{y}\in\Psi_{E_{\mathbf{x}}^{\mathbf{X}}}\setminus\Delta}\mathbf{l}_{\mathbf{xy}}^{\mathbf{XY}}\right) \tag{15}$$

where χ is the set of subsets Δ of $\Psi_{E_{\mathbf{x}}^{\mathbf{X}}}$ that do not justify aggregate atom $\mathsf{op}\{E_{\mathbf{x}}^{\mathbf{X}}\} \prec u$.

Proof. Let $\mathbf{Y}$ be the list of variables occurring in E that do not occur in $\mathbf{X}$. Let $\Delta_I = \{\, \mathbf{y} \in \Psi_{E_{\mathbf{x}}^{\mathbf{X}}} \mid I \models \mathbf{l}_{\mathbf{xy}}^{\mathbf{XY}} \,\}$ and F_I be the formula

$$\bigwedge_{\mathbf{y}\in\Delta_I}\mathbf{l}_{\mathbf{xy}}^{\mathbf{XY}} \to \bigvee_{\mathbf{y}\in\Psi_E\setminus\Delta_I}\mathbf{l}_{\mathbf{xy}}^{\mathbf{XY}}$$

Then, $I \not\models F_I$ and

$$set_{|E/\mathbf{X}|}(\mathbf{x})^I = \{\mathbf{t}_{\mathbf{xy}}^{\mathbf{XY}} \mid \mathbf{y} \in \Delta_I\} = [\Delta_I].$$

Consequently, we have

$$\begin{aligned} I \models (15) &\text{ iff } F_I \text{ is not a conjunctive term of (15)}\\ &\text{iff } \Delta_I \text{ justifies } \mathsf{op}(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u\\ &\text{iff } I \models \mathsf{op}|([\Delta_I]^*) \prec u\\ &\text{iff } I \models \mathsf{op}(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u \end{aligned}$$

□

Lemma 14. *Let $\langle H, I\rangle$ be an agg-ht-interpretation and* op *be an operation name. Then, $\langle H, I\rangle$ satisfies* $\mathtt{op}(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u$ *iff $\langle H, I\rangle$ satisfies*

$$\bigwedge_{\Delta\in\chi}\left(\bigwedge_{\mathbf{y}\in\Delta}\mathbf{l}^{\mathbf{XY}}_{\mathbf{xy}} \rightarrow \bigvee_{\mathbf{y}\in\Psi_{E^{\mathbf{X}}_{\mathbf{x}}}\setminus\Delta}\mathbf{l}^{\mathbf{XY}}_{\mathbf{xy}}\right) \tag{16}$$

where χ is the set of subsets Δ of $\Psi_{E^{\mathbf{X}}_{\mathbf{x}}}$ that do not justify aggregate atom $\mathtt{op}\{E^{\mathbf{X}}_{\mathbf{x}}\} \prec u$.

Proof. Let us denote $\mathtt{op}(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u$ as A in the following.

Case 1: $I \not\models A$. By Lemma 13, it follows that $I \not\models$ (16). Therefore, $\langle H, I\rangle \not\models_{ht} A$ and $\langle H, I\rangle \not\models_{ht}$ (16).

Case 2: $I \models A$. By Lemma 13, it follows that $I \models$ (16). Let

$$\Delta_{\langle H,I\rangle} = \{\, \mathbf{y} \in \Psi_{E^{\mathbf{X}}_{\mathbf{x}}} \mid \langle H, I\rangle \models_{ht} \mathbf{l}^{\mathbf{XY}}_{\mathbf{xy}} \,\}$$

and $F_{\langle H,I\rangle}$ be the formula

$$\bigwedge_{\mathbf{y}\in\Delta_{\langle H,I\rangle}}\mathbf{l}^{\mathbf{XY}}_{\mathbf{xy}} \rightarrow \bigvee_{\mathbf{y}\in\Psi_E\setminus\Delta_{\langle H,I\rangle}}\mathbf{l}^{\mathbf{XY}}_{\mathbf{xy}}$$

Then, $\langle H, I\rangle \not\models F_{\langle H,I\rangle}$ and

$$set_{|E/\mathbf{X}|}(\mathbf{x})^H = \{\mathbf{t}^{\mathbf{XY}}_{\mathbf{xy}} \mid \mathbf{y} \in \Delta_{\langle H,I\rangle}\} = [\Delta_{\langle H,I\rangle}].$$

Consequently, we have

$$\begin{aligned}
\langle H, I\rangle \models (16) \text{ iff } & I \models (16) \text{ and} \\
& F_{\langle H,I\rangle} \text{ is not a conjunctive term of (16)} \\
\text{iff } & F_{\langle H,I\rangle} \text{ is not a conjunctive term of (16)} \\
\text{iff } & \Delta_{\langle H,I\rangle} \text{ justifies } \mathtt{op}(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u \\
\text{iff } & H \models \mathtt{op}|([\Delta_{\langle H,I\rangle}]^*) \prec u \\
\text{iff } & H \models \mathtt{op}(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u \\
\text{iff } & H \models A \\
\text{iff } & \langle H, I\rangle \models A
\end{aligned}$$

□

Lemma 15. *Let Π be a program, $\mathcal{P}$ be the set of all predicate symbols in σ^* other than comparisons, $\mathcal{F}$ be the set of all function symbols corresponding set symbols. Then,*

$$\langle H, I\rangle \models_{ht} \tau\Pi \quad \textit{iff} \quad \langle H, I\rangle \models_{ht} gr_I^{\mathcal{PF}}(\tau^*\Pi)$$

for every agg-ht-interpretation $\langle H, I\rangle$.

Proof. Recall that comparisons are not intensional in the definition of the stable models of a program, that is, they do not belong to $\mathcal{P}$. Then, it is easy to see that $\tau\Pi$ can be obtained

from $gr_I^{\mathcal{PF}}(\tau^*\Pi)$ by replacing each occurrence of $op(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u$, where op is an operation name, by its corresponding formula of the form of (15): Here χ is the set of subsets Δ of $\Psi_{E^{\mathbf{X}}_{\mathbf{x}}}$ that do not justify $op(set_{|E/\mathbf{X}|}(\mathbf{x}))$. Hence, it is enough to show that

$$\langle H, I\rangle \models_{ht} op(set_{|E/\mathbf{X}|}(\mathbf{x}) \prec u) \quad \text{iff} \quad \langle H, I\rangle \models_{ht} (15)$$

This follows from Lemma 15. □

In the following two Lemmas, Π is a program, $\mathcal{P}$ is the set of all predicate symbols in σ^* other than comparisons, $\mathcal{F}$ is the set of all function symbols corresponding set symbols, and let I be an agg-interpretation.

Lemma 16. *I is an FT-agg-stablemodel of $\tau\Pi$ iff it is an FT-agg-stablemodel of $gr_I(\tau^*\Pi)$.*

Proof. By Lemma 15, $\tau\Pi$ and $gr_I(\tau^*\Pi)$ have the same agg-ht-models. □

Lemma 17. *$\mathcal{I}$ is an FT-stable model of $\tau\Pi$ iff I is an agg-stable model of $\tau^*\Pi$.*

Proof. $\mathcal{I}$ is an FT-stable model of $\tau\Pi$
iff I is an FT-agg-stable model of $\tau\Pi$ (Lemma 9)
iff I is an FT-agg-stable model of $gr_I(\tau^*\Pi)$ (Lemma 16)
iff I is an agg-stable model of $\tau^*\Pi$. (Lemma 12)
□

Lemma 18. *A set $\mathcal{A}$ of ground atoms is an FT-stable model of $\tau\Pi$ iff there is some I agg-stable model of $\tau^*\Pi$ such that $\mathcal{A} = Ans(I)$.*

Proof. The *right-to-left* direction follows directly from Lemma 17 by taking noting that there are no comparison symbols in $\tau\Pi$. For the *left-to-right* direction let I be an agg-interpretation such that $p(\mathbf{t})^I$ is true iff $p(\mathbf{t})$ belongs to $\mathcal{A}$. Then, $\mathcal{A}$ is the set of all ground atoms in $\mathcal{I}$ that do not contain comparison symbols and, since $\tau\Pi$ does not contain comparison symbols, it follows that $\mathcal{I}$ is a FT-stable model of $\tau\Pi$ too. The result follows now by Lemma 17. □

Lemma 19. *I is a stable model of* SCA $\cup\, \Gamma$ *iff I is a agg-stable model of* Γ.

Proof. By definition, I is a agg-stable model of Γ iff I is an agg-model of Γ and there is no agg-ht-interpretation $\langle H, I\rangle$ with $H <^{\mathcal{PF}} I$ such that $\langle H, I\rangle$ satisfies Γ iff (Propositions 1 and 2) iff I is a model of SCA $\cup\, \Gamma$ and there is no agg-ht-interpretation $\langle H, I\rangle$ with $H <^{\mathcal{PF}} I$ such that $\langle H, I\rangle$ satisfies SCA $\cup\, \Gamma$ iff I is a stable model of SCA $\cup\, \Gamma$. □

Proof of Theorem 1. Let Π be a program. Then,
$\mathcal{A}$ is a gringo answer set of Π.
iff $\mathcal{A}$ is an FT-stable model of $\tau\Pi$
iff (Lemma 18) there is an agg-stable model I of $\tau^*\Pi$ such that $Ans(I) = \mathcal{A}$
iff (Lemma 19) there is an stable model I of SCA $\cup\, \tau^*\Pi$ such that $Ans(I) = \mathcal{A}$
iff $\mathcal{A}$ is a fo-gringo answer set of Π □

Lemma 20. *$\langle H, I\rangle \models_{dlv} F \leftrightarrow G$ iff*

- $I \models F \leftrightarrow G$, *and*
- $H \models F$ *and* $I \models F$ *iff* $H \models G$ *and* $I \models G$.

Proof. $\langle H, I\rangle \models_{dlv} F \leftrightarrow G$
iff $\langle H, I\rangle \models_{dlv} F \rightarrow G$ and $\langle H, I\rangle \models_{dlv} G \rightarrow F$
iff both

- $I \models F \rightarrow G$ and either $H \not\models F$ or $I \not\models F$ or both $H \models G$ and $I \models G$, and
- $I \models G \rightarrow F$ and either $H \not\models G$ or $I \not\models G$ or both $H \models F$ and $I \models F$;

iff

- $I \models F \rightarrow G$ and $I \models G \rightarrow F$, and
- either $H \not\models F$ or $I \not\models F$ or both $H \models G$ and $I \models G$, and
- either $H \not\models G$ or $I \not\models G$ or both $H \models F$ and $I \models F$;

iff

- $I \models F \leftrightarrow G$, and
- $H \models F$ and $I \models F$ implies $H \models G$ and $I \models G$, and
- $H \models G$ and $I \models G$ implies $H \models F$ and $I \models F$;

iff

- $I \models F \leftrightarrow G$, and
- $H \models F$ implies $H \models G$, and
- $H \models G$ implies $H \models F$;

iff

- $I \models F \leftrightarrow G$, and
- $H \models F \leftrightarrow G$. □

Lemma 21. *Let E be an aggregate element of the form (5) with free variables $\mathbf{X}$ and bound variables $\mathbf{Y}$ and let $\langle H, I\rangle$ be a standard ht-interpretation. Let $\mathbf{x}$ be a list of ground terms of sort s_{prg} of the same length as $\mathbf{X}$, d_{tuple} a domain element of sort tuple, $l'_i = (l_i)^{\mathbf{X}}_{\mathbf{x}}$ and $t'_i = (t_i)^{\mathbf{X}}_{\mathbf{x}}$. Then, $\langle H, I\rangle$ dlv-staisfies (13) iff the the following two conditions are equivalent:*

1. *d_{tuple} belongs to $set_{|E|}(\mathbf{x})^I$,*
2. *there is a list $\mathbf{c}$ of domain elements of sort s_{prg} of the same length as $\mathbf{Y}$ such that $d_{tuple} = \langle (t''_1)^I, \ldots, (t''_m)\rangle^I$ and I satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $t''_i = (t'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $l''_i = (l'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $\mathbf{y} = \mathbf{c}^*$,*

and the the following two conditions are aslo equivalent:

3. *d_{tuple} belongs to $set_{|E|}(\mathbf{x})^H$,*
4. *there is a list $\mathbf{c}$ of domain elements of sort s_{prg} of the same length as $\mathbf{Y}$ such that $d_{tuple} = \langle (t''_1)^I, \ldots, (t''_m)\rangle^H$ and H satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $t''_i = (t'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $l''_i = (l'_i)^{\mathbf{Y}}_{\mathbf{y}}$ and $\mathbf{y} = \mathbf{c}^*$.*

Proof. From Lemma 20, $\langle H, I\rangle$ dlv-staisfies (13) iff both H and I satisfy (13). The result follows now by Lemma 1. □

Proof of Proposition 3. Pick any list $\mathbf{c}$ of domain elements of sort s_{prg} and let $\mathbf{x} = \mathbf{c}^*$. Then, the result follows directly by Lemma 21. □