Journal of Artificial Intelligence Research 80 (2024) 977-1031 Submitted 12/2023; published 07/2024

Axiomatization of Non-Recursive Aggregates in First-Order Answer Set Programming

Jorge Fandinno JFANDINNO@UNOMAHA.EDU
Zachary Hansen ZACHHANSEN@UNOMAHA.EDU
Yuliya Lierler YLIERLER@UNOMAHA.EDU
University of Nebraska Omaha, PKI 280D,
Omaha, NE 68182 USA

Abstract

This paper contributes to the development of theoretical foundations of answer set programming. Groundbreaking work on the SM operator by Ferraris, Lee, and Lifschitz proposed a definition/semantics for logic (answer set) programs based on a syntactic transformation similar to parallel circumscription. That definition radically differed from its predecessors by using classical (second-order) logic and avoiding reference to either grounding or fixpoints. Yet, the work lacked the formalization of crucial and commonly used answer set programming language constructs called *aggregates*. In this paper, we present a characterization of logic programs with aggregates based on a many-sorted generalization of the SM operator. This characterization introduces new function symbols for aggregate operations and aggregate elements, whose meaning can be fixed by adding appropriate axioms to the result of the SM transformation. We prove that our characterization coincides with the `ASP-Core-2` semantics for logic programs and, if we allow non-positive recursion through aggregates, it coincides with the semantics of the answer set solver CLINGO.

1. Introduction

Answer set programming (ASP) is a form of declarative logic programming that was originally developed to address the challenges of non-monotonic reasoning arising in artificial intelligence (Lifschitz, Schaub, & Woltran, 2018). Ever since its inception, it has been an important player in knowledge representation and reasoning. For example, ASP's use of negation-as-failure provides a natural solution to the historically challenging frame problem, allowing programmers to express laws of inertia with just a couple of rules. As a knowledge representation and reasoning paradigm, it is particularly well-suited to solving knowledge-intensive search and optimization problems (Lifschitz, 2019). It has been fruitfully applied to numerous challenging domains, such as space shuttle decision support systems (Balduccini & Gelfond, 2005; Balduccini, Gelfond, Nogueira, Watson, & Barry, 2001), complex scheduling (Abels, Jordi, Ostrowski, Schaub, Toletti, & Wanko, 2021; Balduccini, 2011; Ricca, Grasso, Alviano, Manna, Lio, Iiritano, & Leone, 2012), planning (Cao Tran, Pontelli, Balduccini, & Schaub, 2023), robotics (Aker, Erdogan, Erdem, & Patoglu, 2011; Gebser, Obermeier, Otto, Schaub, Sabuncu, Nguyen, & Son, 2018), and adaptive Linux package configuration (Gebser, Kaminski, & Schaub, 2011). The success of these applications relies on the combination of rich knowledge representation languages with effective solvers. One of the most useful constructs of the ASP language are *aggregates*: intuitively, these are functions that apply to sets. The semantics of aggregates have been extensively studied in the literature (Simons, Niemelä, & Soininen, 2002; Dovier, Pontelli, & Rossi, 2003; Pelov, Denecker, & Bruynooghe, 2007; Son &

Pontelli, 2007; Ferraris, 2011; Faber, Pfeifer, & Leone, 2011; Gelfond & Zhang, 2014, 2019; Cabalar, Fandinno, Schaub, & Schellhorn, 2019). 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 one, then the semantics of the propositional program is defined. However, this makes reasoning about programs with variables cumbersome. For instance, it makes it difficult to consider the rules of a program modeling some problem in separation from a specific instance of that problem (as the instance provides the grounding context). The characterization of programs with aggregates introduced in this paper overcomes this obstacle. Recent work by Fandinno, Hansen, and Lierler (2022a) illustrates the importance of this achievement by applying the introduced semantics to the realm of formal verification of programs with aggregates. In particular, the authors used the presented semantics to argue the correctness of logic programs with aggregates representing encodings of the Graph Coloring and the Traveling Salesman problems. In the proofs, no reference to the specific instances of the problems is used.

We base our characterization of aggregates on the `ASP-Core-2` standard, which is a widely accepted, foundational fragment of ASP modeling languages (Calimeri, Faber, Gebser, Ianni, Kaminski, Krennwallner, Leone, Maratea, Ricca, & Schaub, 2020). This standard does not permit recursion through aggregates. Modern solvers such as CLINGO (Gebser, Kaminski, Kaufmann, & Schaub, 2019) and DLV (Adrian, Alviano, Calimeri, Cuteri, Dodaro, Faber, Fuscà, Leone, Manna, Perri, Ricca, Veltri, & Zangari, 2018) do not have this restriction. Yet, these solvers may compute distinct answers for the same input program — in particular, CLINGO and DLV diverge in their interpretation of recursive aggregates. The solver CLINGO (Gebser et al., 2019) follows the `Abstract Gringo` semantics (Gebser, Harrison, Kaminski, Lifschitz, & Schaub, 2015) for recursive aggregates. We show that if we consider non-positive recursion through aggregates, then our characterization also agrees with the `Abstract Gringo` semantics implemented by CLINGO. In this paper, we do not address unrestricted recursion through aggregates, as our main goal is to axiomatize aggregates as understood in the `ASP-Core-2` semantics (Calimeri et al., 2020). The fact that for programs with aggregates with non-positive recursion our characterization agrees with the `Abstract Gringo` semantics came at no cost to the complexity of the proposed approach. On the other hand, considering unrestricted recursion through aggregates will require either an introduction of restrictions on the type of studied aggregates (as Lifschitz, 2022 does) or a more complex translation that is unnecessary to achieve our main goal.

The following is a summary of the contributions of this paper that can also be used as a road map for the structure of the paper. Before proceeding to the main parts of the paper we would like to position this work properly within the body of related work. Thus, we begin the presentation with a section devoted to related work. We then review the preliminaries that include (i) the precise definition for the syntax of the kinds of programs considered in this work and (ii) operator SM for many-sorted signatures. Next, we introduce a translation from logic programs to many-sorted first-order logic and define the semantics of aggregates using the SM operator that yields a second-order formula (Section 4). We establish the conditions on the models of our translation that provide us with the semantics of a logic program. This provides a meta-logical overview of our formalism (Section 4). We prove that our semantics coincides with the `ASP-Core-2` semantics and that when non-positive recursion is allowed it coincides with the `Abstract Gringo` semantics (Section 5). We present an axiomatization of aggregates and prove that it correctly captures the conditions imposed in Section 4 on the models of the translation. We derive our Main Theorem, which states that answer sets according to the `ASP-Core-2` semantics correspond to the models of

the formula obtained from our translation that satisfy the mentioned axioms (Section 6). We show that under certain conditions, we can replace the second-order formalization by a first-order one. In particular, we show that for programs with aggregates that apply to finite sets, we can replace the second-order axiomatization of aggregates by a first-order one (Section 7). It is important to note that the restriction to finite aggregates is not a practical limitation as solvers cannot process infinite sets. We provide rigorous and detailed proofs of our results in Appendix A. These proofs are in of themselves a key contribution — they provide a deeper understanding of aggregates within our formalism, and they lay a foundation for future extensions.

Parts of this paper appeared in the Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (Fandinno, Hansen, & Lierler, 2022b). The presentation of this paper extends earlier results to new kinds of aggregates, namely, `sum+`, `min`, and `max`; includes details on the kinds of logic programs with aggregates that can be captured by the so-called completion transformation, which allows the use of first-order logic in place of the SM operator transformation that results in second-logic formalization; and includes a thorough, comprehensive account of all formal arguments and required background.

2. Related Work

This section discusses the main differences between our proposed formalism and previous approaches to defining aggregate semantics without reference to grounding.

Classical Logic Language. Lee, Lifschitz, and Palla (2008) translate certain count aggregates with a ground guard into an existentially quantified first-order formula. This approach is inapplicable to more general count aggregates and to sum aggregates, which occur commonly in practice. Similarly to our approach, Lifschitz (2022) employs an extension of the τ^* translation (Fandinno, Lifschitz, Lühne, & Schaub, 2020) to obtain first-order formula representations of programs; this approach allows some count aggregates with a non-ground guard. However, it is not applicable to either count aggregates with an inequality comparison or to sum aggregates. Conversely, our first-order characterization has no such restrictions and can accommodate all types of aggregates defined in the `ASP-Core-2` semantics (Calimeri et al., 2020).

Language Extensions. Bartholomew, Lee, and Meng (2011), Ferraris and Lifschitz (2010), and Lee and Meng (2012) modify the SM operator (Ferraris et al., 2011) to handle aggregates by extending the language of second-order logic with new expressions that are used to capture aggregate expressions. Ferraris and Lifschitz (2010) and Lee and Meng (2012) employ the language of second-order logic extended with generalized quantifiers (Peters & Westerstahl, 2006), while Bartholomew et al. (2011) extend the logic with a specific aggregate construction. In a similar vein, Asuncion, Chen, Zhang, and Zhou (2015) also augment the SM operator by extending the language of second-order logic with aggregate atoms as new syntactic constructs. They show that for finite structures, their semantics can be characterized by a first-order formula extended with the new aggregate construct. Vanbesien, Bruynooghe, and Denecker (2022) define the semantics by applying approximation fixpoint theory to first-order logic extended with a new aggregate construct.

In contrast to our approach, these formalisms rely on extensions of the language of classical logic with special treatment for new syntactic constructs associated with aggregates. As a result, they cannot leverage the whole body of results that exist for classical logic, something that our approach allows. This is important for several reasons. Arguably, the classical logic fragment is better

understood and studied so that we can rely on the body of earlier work in analyzing logic programs. For instance, there is the potential to use existing first-order theorem provers for verifying properties of certain programs (Fandinno et al., 2020) and equivalences between programs (Fandinno, Hansen, Lierler, Lifschitz, & Temple, 2023).

Quantified Equilibrium Logic Cabalar, Fandinno, Fariñas del Cerro, and Pearce (2018) introduce intensional sets as first-class citizens into Quantified Equilibrium Logic (Pearce & Valverde, 2005) with partial functions (Cabalar, 2011) and provide a formalization of aggregates. This corresponds to a useful intuition that aggregates are functions that apply to sets. In the works by Ferraris and Lifschitz (2010), Bartholomew et al. (2011), Lee and Meng (2012), Lee et al. (2008), Lifschitz (2022), Asuncion et al. (2015), and Vanbesien et al. (2022) this idea is not explicitly represented in the target language. Similar to the work by Cabalar et al. (2018), our approach provides a direct formalization of the idea that aggregates are functions that apply to sets, but it aims to exclusively use the language of classical logic (instead of adding intensional sets as a new construct in the language). As stated above this is important for leveraging the whole body of results of the logic.

It is also worth mentioning that, but for the work by Lifschitz (2022), none of the above approaches have formally related their semantics to the `ASP-Core-2` semantics. Thus, it is an open question whether they actually capture the language used in practice.

3. Preliminaries

In this section, we review the syntax of programs with aggregates (Section 3.1) and the SM operator (Ferraris et al., 2011) extended to the many-sorted case (Section 3.3).

3.1 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, $\overline{n} < c$[1].

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*:

$$= \quad < \quad > \quad \leq \quad \geq \tag{1}$$

1. Note how there are no restrictions on how symbolic constants are ordered with respect to each other, apart from the fact that they are ordered.

Listing 1: Encoding of the graph coloring problem using the basic ASP language.

```
assign(X,Z) :- vertex(X), color(Z), not not assign(X,Z).
:- vertex(X), not #count{ X,Z : assign(X,Z), color(Z) } = 1.
:- edge(Y,X), assign(Y,Z), assign(X,Z).
```

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*. We consider an expression $t \neq t'$ to be an abbreviation for basic literal *not* $t = t'$.

An *aggregate element* has the form

$$t_1, \ldots, t_k : l_1, \ldots, l_m \tag{2}$$

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 an expression of the form

$$\texttt{\#op}\{E\} \prec u \tag{3}$$

where op is an operation name, E is an aggregate element (2), $\prec$ is one of the comparison symbols in (1), and u is a program term, called the *guard*. We consider operation names[2]

```
count   sum   sum+   min   max.
```

For example, expression

$$\texttt{\#sum}\{K,X,Y : in(X,Y), cost(K,X,Y)\} > J$$

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

$$Head \texttt{ :- } B_1, \ldots, B_n, \tag{4}$$

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 n is 0 we call a rule a *fact* and may drop the rule operator. When the head of the rule is an atom we call the rule *normal*. A *program* is a finite set of rules. The program in Listing 1 encodes the graph coloring problem with this syntax.

A *choice rule* is an expression of the form

$$\{A_0 : A_1, \ldots, A_k\} \prec u \texttt{ :- } B_1, \ldots, B_n. \tag{5}$$

2. Operation sum+ is not part of the ASP-Core-2 language, but it is part of the Abstract Gringo language. We include it here to extend our comparison to the latter.

Listing 2: Encoding of the graph coloring problem using a choice rule.

```
{ assign(X,Z) : color(Z) } = 1 :- vertex(X).
:- edge(Y,X), assign(Y,Z), assign(X,Z).
```

where each A_i is an atomic formula, each B_i is a literal, $\prec$ is a comparison symbol and u is a numeral; it is understood as an abbreviation for the following pair of rules

$$A_0 \text{ :- } A_1, \ldots A_k, B_1, \ldots, B_n, \mathit{not}\ \mathit{not}\ A_0. \tag{6}$$

$$\text{:- } B_1, \ldots, B_n, \mathit{not}\ \texttt{\#count}\{\mathbf{t} : A_0, A_1, \ldots, A_k\} \prec u. \tag{7}$$

where $\mathbf{t}$ is a list of program terms such that A_0 is of the form $p(\mathbf{t})$ for some symbolic constant p. As usual, we allow that "$\prec u$" or "$: A_1, \ldots A_k$," (or both of them) are omitted from choice rules. If "$\prec u$" is omitted, then (7) is omitted; and if "$: A_1, \ldots A_k$" is omitted, then $A_1, \ldots A_k$ is omitted from (6) and (7).

Usually, the use of choice rules allows for a more compact and readable encoding of a problem. For example, the program in Listing 2 contains a choice rule and is an abbreviation for the program in Listing 1 (the last rule in both of these programs coincide).

Aggregate operations Each operation name op is associated with a function $\widehat{\texttt{op}}$ 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,

- $\widehat{\texttt{count}}(\Delta)$ is the numeral corresponding to the cardinality of Δ, if Δ is finite; and *sup* otherwise.
- $\widehat{\texttt{sum}}(\Delta)$ 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.[3] If Δ is empty, then $\widehat{\texttt{sum}}(\Delta) = 0$.
- $\widehat{\texttt{sum+}}(\Delta)$ is the numeral corresponding to the sum of the weights of all tuples in Δ whose weights are positive, if Δ contains finitely many such tuples; and *sup* otherwise.
- $\widehat{\texttt{min}}(\Delta)$ is *sup* if Δ is empty, is *inf* if Δ is infinite, or is the least element of the set consisting of the first members of elements of Δ if Δ is finite and non-empty.
- $\widehat{\texttt{max}}(\Delta)$ is *inf* if Δ is empty, is *sup* if Δ is infinite, or is the greatest element of the set consisting of the first members of elements of Δ if Δ is finite and non-empty.

Recursion over aggregates In proposing the new definition of the semantics of programs with aggregates, *we will assume that programs do not have positive recursion over aggregates*. As such, this kind of programs will lead the discourse of this paper. This is a less restrictive assumption than the one used in the ASP-Core-2 semantics (Calimeri et al., 2020), which requires that programs have neither positive nor negative recursion over aggregates.

We now define the concept of recursion over aggregates. A *predicate symbol* is a pair p/n, where p is a symbolic constant and n is a nonnegative integer. About a program or another syntactic expression, we say that a predicate symbol p/n *occurs* in it if it contains an atom of the form $p(t_1, \ldots, t_n)$. For a program Π, its *(directed predicate) dependency graph* is defined by

3. 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 (Gebser et al., 2015).

1. a set of vertices containing all predicate symbols occurring in Π,

2. a set of edges containing an edge (h,b) for every normal rule (4) with h being the predicate symbol occurring in the atom *Head* of the rule and b being any predicate symbol in one of the literals $B_1, \ldots, B_n$ of the rule.

We say that an occurrence of an aggregate literal B_i in a rule R, of the form (4), is *recursive* with respect to a program Π containing R if there is a path from some predicate symbol occurring in B_i to the predicate symbol occurring in *Head* in the dependency graph of Π (Harrison & Lifschitz, 2018). We say that a program Π has *recursion over aggregates* if there is an occurrence of an aggregate literal in a rule of Π that is recursive with respect to Π.

We say that an occurrence of a predicate symbol p/n or any other expression is *strictly positive* if it is not in the scope of negation. For example, literals

$$\begin{array}{l} \mathit{not}\ r(X,Y,Z) \\ \texttt{\#sum}\{Y,Z : \mathit{not}\ r(X,Y,Z)\} \geq 0 \\ \mathit{not}\ \mathit{not}\ \texttt{\#sum}\{Y,Z : r(X,Y,Z)\} \geq 0 \end{array}$$

contain no strictly positive occurrence of the predicate symbol $r/3$. For a program Π, its *positive (directed predicate) dependency graph* is defined by

1. a set of vertices containing all predicate symbols occurring in Π ,

2. a set of edges containing an edge (h,b) for every normal rule (4) with h being the predicate symbol occurring in the atom *Head* of the rule and b being any predicate symbol that has a strictly positive occurrence in one of the literals $B_1, \ldots, B_n$ of the rule.

We say that an occurrence of a strictly positive aggregate literal B_i in the body of rule R, of the form (4), is *positively recursive* with respect to a program Π containing R if there is a path from some predicate symbol with a strictly positive occurrence in B_i to the predicate symbol occurring in *Head* in the positive dependency graph of Π. We say that a program Π has *positive recursion over aggregates* if there is an occurrence of an aggregate literal in a rule of Π that is positively recursive with respect to Π. Clearly every program with positive recursion over aggregates has recursion over aggregates. The converse may not be the case. For instance, a program containing any of the following rules:

$$\begin{array}{l} r(X,Y,Z) \text{ :- } q(X,Y,Z), \texttt{\#sum}\{Y,Z : \mathit{not}\ r(X,Y,Z)\} \geq 0 \\ r(X,Y,Z) \text{ :- } q(X,Y,Z), \mathit{not}\ \texttt{\#sum}\{Y,Z : r(X,Y,Z)\} \geq 0 \end{array}$$

has recursion over aggregates but no positive recursion; a program containing a rule

$$r(X,Y,Z) \text{ :- } q(X,Y,Z), \texttt{\#sum}\{Y,Z : r(X,Y,Z)\} \geq 0$$

has an aggregate with positive recursion.

3.2 Background on Many-Sorted Second-Order Logic

We start by recalling the standard definition of many-sorted first-order logic. We restate several definitions here for clarity. The logical primitives consist of *binary connectives* ($\wedge, \vee, \rightarrow$), *0-place connectives* truth and falsity ($\top, \bot$), and the *quantifiers* ($\forall, \exists$). Negation and equivalence are assumed to be abbreviations: $\neg F$ stands for $F \rightarrow \bot$, and $F \leftrightarrow G$ stands for $(F \rightarrow G) \wedge (G \rightarrow F)$. A many-sorted signature σ 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. We define the *arity*[4] of every function or predicate constant as an n-tuple of sorts ($n \geq 0$ for predicate symbols and $n \geq 1$ for function symbols). For the sake of clarity, we write $s_1 \times \cdots \times s_n$ instead of $\langle s_1, \ldots, s_n \rangle$ when referring to the arity of predicates and $s_1 \times \cdots \times s_n \rightarrow s_{n+1}$ instead of $\langle s_1, \ldots, s_n, s_{n+1} \rangle$ when referring to the arity of functions (here, s_{n+1} is the *value sort* of the function constant). A predicate constant whose arity is the empty tuple is called a *proposition*. A function constant whose arity consists only of a value sort s is called an *object constant*.

We assume that there are infinitely many object variables for each sort and infinitely many predicate variables for each arity $s_1 \times \cdots \times s_n$.

A *term* is defined recursively, as follows:

- object constants and variables of sort s are terms of sort s,
- if a function constant f has arity $s_1 \times \cdots \times s_n \rightarrow s_{n+1}$ and each t_i is a term of a subsort of s_i ($1 \leq i \leq n$), then $f(t_1, \ldots, t_n)$ is a term of sort s_{n+1}.

A *atomic formula* is an expression of the form

- $t_1 = t_2$, where t_1 and t_2 are terms that share a common supersort, or
- $p(t_1, \ldots, t_n)$, where p is a predicate constant or predicate variable of arity $s_1 \times \cdots \times s_n$ and each t_i ($1 \leq i \leq n$) is a term of a subsort of s_i.

We typically use infix notation for writing atomic formulas whose predicate symbol is a relation (1). A *formula* is defined recursively:

- atomic formulas are formulas,
- both 0-place connectives are formulas,
- for any binary connective $\odot$, if F and G are formulas, then $F \odot G$ is a formula,
- for any quantifier Q, if F is a formula and x is a variable, then QxF is a formula.

Example 1. *Consider a signature σ with sorts s_1 and s_2 (s_2 is a subsort of s_1), function constants f of arity $s_1 \rightarrow s_2$ and c of arity s_1, and predicate constant p of arity $s_1 \times s_2$. Then, c, $f(c)$, and $f(f(c))$ are terms, and $c = f(c)$, $p(c, f(c))$ are atomic formulas.*

An occurrence of a variable x in a formula F is *bound* if it occurs in a part of F of the form QxG with Q being a quantifier; otherwise, it is *free* in F. An *interpretation* I of a signature σ consists of the following:

4. This definition of an arity is consistent with lecture notes on *Many-Sorted Logic* by Galogero G. Zabra (2006).

F, G	$F \wedge G$	$F \vee G$	$F \to G$
true, **true**	**true**	**true**	**true**
true, **false**	**false**	**true**	**false**
false, **true**	**false**	**true**	**true**
false, **false**	**false**	**false**	**true**

Table 1: Truth values of the binary connectives.

- a nonempty universe $|I|^s$ for each sort s,
- for each function constant f of arity $s_1 \times \cdots \times s_n \to s_{n+1}$, a function $f^I : |I|^{s_1} \times \cdots \times |I|^{s_n} \to |I|^{s_{n+1}}$,
- for each proposition p, p^I is either **true** or **false**,
- for each predicate constant p of arity $s_1 \times \cdots \times s_n$ with $n \geq 1$, a subset $p^I \subseteq |I|^{s_1} \times \cdots \times |I|^{s_n}$.

When sort s_1 is a subsort of sort s_2, an interpretation additionally satisfies the condition $|I|^{s_1} \subseteq |I|^{s_2}$.

Names. Given an interpretation I of a signature σ, we extend the signature with *names* for domain elements and relations as follows.

- For every universe $|I|^s$, take every distinct element $\zeta \in |I|^s$ and select a fresh symbol ζ^* as the name of ζ.
- For every relation ρ formed as a subset of every possible Cartesian product $|I|^{s_1} \times \cdots \times |I|^{s_n}$ given sorts $s_1, \ldots, s_n$, select a fresh symbol ρ^* as the name for this relation.

The signature σ^I is obtained from σ by adding all names ζ^* as object constants of the corresponding sort as well as names ρ^* to the set of predicate constants. The interpretation I can be extended to the new signature σ^I by defining $(\zeta^*)^I = \zeta$ for all $\zeta \in |I|^s$ and mapping each symbol ρ^* to its respective relation ρ.

Semantics. For any term t of σ^I that does not contain variables, we recursively define the element t^I of the universe that is assigned to t by I. If t is an object constant then t^I is part of the interpretation I. For other terms, t^I is defined by the equation

$$f(t_1, \ldots, t_n)^I = f^I(t_1^I, \ldots, t_n^I)$$

for all function constants f of arity $s_1 \times \cdots \times s_n \to s_{n+1}$ with $n > 0$. The truth value – **true** or **false** – assigned to sentence F by interpretation I is defined as follows:

- $(t_1 = t_2)^I =$ **true** iff t_1^I is the same as t_2^I.
- $p(t_1, \ldots, t_n)^I =$ **true** iff $(t_1^I, \ldots, t_n^I)$ belongs to p^I
- $\bot^I =$ **false**, $\top^I =$ **true**
- $(F \odot G)^I = F^I \odot G^I$ for every binary connective $\odot$ where binary connectives are defined in Table 1.

- $\forall x F(x)^I = \mathbf{true}$ iff x is a variable of sort s and $F(\zeta^*)^I = \mathbf{true}$ for all $\zeta \in |I|^s$.
- $\exists x F(x)^I = \mathbf{true}$ iff x is a variable of sort s and $F(\zeta^*)^I = \mathbf{true}$ for some $\zeta \in |I|^s$.
- $\forall v F(v)^I = \mathbf{true}$ iff v is a predicate variable of arity $s_1 \times \cdots \times s_n$ and $F(\rho^*)^I = \mathbf{true}$ for all relations ρ in $|I|^{s_1} \times \cdots \times |I|^{s_n}$.
- $\exists v F(v)^I = \mathbf{true}$ iff v is a predicate variable of arity $s_1 \times \cdots \times s_n$ and $F(\rho^*)^I = \mathbf{true}$ for some relation ρ in $|I|^{s_1} \times \cdots \times |I|^{s_n}$.

Satisfaction and Models. An interpretation I over signature σ *satisfies* formula F, written $I \models F$, if $F^I = \mathbf{true}$. An interpretation I over signature σ satisfies the set of formulas Γ if $I \models F$ for all $F \in \Gamma$. An interpretation I is called a model of F (resp. Γ) if it satisfies F (resp. Γ).

3.3 Operator SM for Many-sorted Signatures

Ferraris et al. (2011) proposed the operator SM as a formal tool for studying the semantics of logic programs. In their work, a logic program is first identified with a first-order sentence called the *formula representation* of the program. Second, the SM operator is applied to this sentence, transforming it into a second-order logic formula. Models of the resulting formula capture the stable models of the program's formula representation (and hence, the logic program's stable models). Our goal is to generalize this approach to the case of logic programs with aggregates. For this, we found it essential to invoke many-sorted logic. In particular, we develop an approach in which a logic program with aggregates is identified with a many-sorted first-order sentence. Then, as in pioneering work by Ferraris et al. (2011), the SM operator is applied to this formula representation to obtain stable models of the associated logic program. Here, we review the generalization of the SM operator to the many-sorted case.

The definition of the operator SM for a many-sorted signature is a straightforward generalization of the unsorted case presented by Ferraris et al. (2011). If p and u are predicate constants or variables of the same arity (note that while the original definition does not account for sort information, here arity refers to both number and sort of the arguments), then $u \leq p$ stands for the formula

$$\forall \mathbf{W}(u(\mathbf{W}) \rightarrow p(\mathbf{W})),$$

where $\mathbf{W}$ is a tuple of distinct object variables matching the arity of p and u. If $\mathbf{p}$ and $\mathbf{u}$ are tuples $p_1, \ldots, p_n$ and $u_1, \ldots, u_n$ of predicate constants or variables such that each p_i and u_i have the same arity, then $\mathbf{u} \leq \mathbf{p}$ stands for the conjunction

$$(u_1 \leq p_1) \wedge \cdots \wedge (u_n \leq p_n),$$

and $\mathbf{u} < \mathbf{p}$ stands for $(\mathbf{u} \leq \mathbf{p}) \wedge \neg(\mathbf{p} \leq \mathbf{u})$. For any many-sorted first-order formula F and a list $\mathbf{p}$ of predicate constants, by $\mathrm{SM}_{\mathbf{p}}[F]$ we denote the second-order formula

$$F \wedge \neg \exists \mathbf{u}\big((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})\big)$$

where $\mathbf{u}$ is a list of distinct predicate variables $u_1, \ldots, u_n$ of the same length as $\mathbf{p}$, such that the arity of each u_i is the same as the arity of p_i, and $F^*(\mathbf{u})$ is defined recursively:

- $F^* = F$ for any atomic formula F that does not contain members of $\mathbf{p}$,

- $p_i(\mathbf{t})^* = u_i(\mathbf{t})$ for any predicate symbol p_i belonging to $\mathbf{p}$ and any list $\mathbf{t}$ of terms,
- $(F \wedge G)^* = F^* \wedge G^*$
- $(F \vee G)^* = F^* \vee G^*$
- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$
- $(\forall x F)^* = \forall x F^*$
- $(\exists x F)^* = \exists x F^*$

If the list $\mathbf{p}$ is empty, then we understand $\mathrm{SM}_{\mathbf{p}}[F]$ as F. For a finite theory Γ, we write $\mathrm{SM}_{\mathbf{p}}[\Gamma]$ to represent $\mathrm{SM}_{\mathbf{p}}[F]$, where F is the conjunction of all formulas in Γ.

4. Programs with Aggregates as Many-Sorted First-Order Sentences

We start this section by presenting the details of a translation that allows us to identify a logic program with aggregates with a many-sorted first-order sentence. We then use the SM operator applied to these formulas to provide the semantics of these logic programs.

4.1 From Programs to First-order Sentences of Two Sorts

We present the translation κ that turns a program Π (whose aggregates lack positive recursion) into a first-order sentence with equality over a signature σ_Π of *two sorts*[5]. To define this signature, we first introduce the concepts of a global variable and a 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.

For instance, in rule

$$p(X) \text{ :- } q(X), \texttt{\#sum}\{Y,Z : r(X,Y,Z)\} \geq 1. \tag{8}$$

the only global variable is X.

A *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. We say that $E/\mathbf{X}$ 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. For instance,

$$Y,Z : r(X,Y,Z)/X$$

is the only set symbol occurring in rule (8). We say that $E/\mathbf{X}$ occurs in a program if $E/\mathbf{X}$ occurs in some rule of the program. For the sake of readability, we associate each set symbol $E/\mathbf{X}$ with a different name $|E/\mathbf{X}|$.

5. This translation is similar to the τ^* translation by Fandinno et al. (2020) in that it translates a fragment of the language of logic programs into first-order sentences of two sorts. Our translation aims at a different fragment though: logic programs with aggregates rather than logic programs with arithmetic expressions. For the sake of simplicity, we leave for future work on how to combine these two orthogonal features of logic programs.

Predicate or function symbol	Arity
$>, \geq, <, \leq$	$s_{prg} \times s_{prg}$
count, *sum*, sum^+, *min*, *max*	$s_{set} \to s_{prg}$
$set_{\lvert E/\mathbf{X}\rvert}$	$s_{prg} \times \cdots \times s_{prg} \to s_{set}$

Table 2: Predicate and function symbols of the signature σ_Π. The function symbols $set_{|E/\mathbf{X}|}$ take as many arguments as the number of global variables in $\mathbf{X}$.

As stated earlier, the signature σ_Π is defined over *two sorts*. The first sort is called the *program sort*; all program terms are of this sort. The second sort is called the *set sort*; it contains entities that are *sets* (of tuples of object constants of the program sort). We denote the two sorts in an intuitive manner: s_{prg} and s_{set}. For a program Π, signature σ_Π contains:

a. all ground terms as object constants of the program sort;

b. all predicate symbols occurring in Π as predicate constants with all arguments of sort program;

c. the comparison symbols other than equality as predicate constants of arity $s_{prg} \times s_{prg}$;

d. function constants *count*, *sum*, sum^+, *min*, *max* of arity $s_{set} \to s_{prg}$;

e. a function constant $set_{|E/\mathbf{X}|}$ of arity $s_{prg} \times \cdots \times s_{prg} \to s_{set}$ for each set symbol $E/\mathbf{X}$ occurring in program Π. This function symbol takes as many arguments of the program sort as there are variables in $\mathbf{X}$. If $\mathbf{X}$ is the empty list, then $set_{|E/\mathbf{X}|}$ is an object constant of sort set.

Table 2 summarizes the bullets above. Intuitively, the result of *count* is the cardinality of the set passed as an argument. The result of *sum* is the sum of weights of all elements from the set passed as an argument; similarly, the result of sum^+ is the sum of all positively weighted elements from the set. The result of *min* is the least element from the tuples' first positions, and *max* is the greatest. Finally, $set_{|E/\mathbf{X}|}(t_1,\ldots,t_k)$ represents the set of elements corresponding to the aggregate element E once all global variables in $\mathbf{X} = X_1,\ldots,X_k$ are replaced by ground terms $t_1,\ldots,t_k$. We formalize these claims below. As customary in arithmetic, we use infix notation in constructing atoms that utilize predicate symbols $>, \geq, <, \leq$. Expression $t_1 \neq t_2$ is considered an abbreviation for the formula $\neg(t_1 = t_2)$. In the following, we use letters X, Y, Z and their variants to denote variables of sort s_{prg} and letter S and its variants to denote variables of sort s_{set}. We use their boldface variants to denote lists of variables of that sort.

We now describe a translation κ 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 $\kappa_\mathbf{Z}$ for all elements of R as follows:

- for every atomic formula A occurring outside of an aggregate literal, its translation $\kappa_\mathbf{Z} A$ is A itself; $\kappa_\mathbf{Z} \bot$ is $\bot$;
- for an aggregate atom A of the form $\mathtt{\#op}\{E\} \prec u$, its translation $\kappa_\mathbf{Z}$ is the atom

$$op(set_{|E/\mathbf{X}|}(\mathbf{X})) \prec u$$

where $\mathbf{X}$ is the list of variables in $\mathbf{Z}$ occurring in E;

- for every (basic or aggregate) literal of the form *not A* its translation $\kappa_{\mathbf{Z}}(\mathit{not}\ A)$ is $\neg\kappa_{\mathbf{Z}}A$; for every literal of the form *not not A* its translation $\kappa_{\mathbf{Z}}(\mathit{not\ not}\ A)$ is $\neg\neg\kappa_{\mathbf{Z}}A$.

We now define the translation κ as follows:

- for every rule R of form (4), its translation κR is the universal closure of the implication

$$\kappa_{\mathbf{Z}}B_1 \wedge \cdots \wedge \kappa_{\mathbf{Z}}B_n \rightarrow \kappa_{\mathbf{Z}}\mathit{Head},$$

  where $\mathbf{Z}$ is the list of the global variables of R.

- for every program Π, its translation $\kappa\Pi$ is the first-order theory containing κR for each rule R in Π.

Example 2. *The result of applying κ to a program consisting of rule* (8) *and the rules*

$$s(X) \text{ :- } q(X), \#sum\{Y : r(X,Y,Z)\} \geq 1. \tag{9}$$
$$t \text{ :- } \#sum\{Y,Z : r(X,Y,Z)\} \geq 1. \tag{10}$$
$$q(a).\ q(b).\ q(c). \tag{11}$$
$$r(a,1,a).\ r(b,-1,a).\ r(b,1,a).\ r(b,1,b).\ r(c,0,a). \tag{12}$$

is the first-order theory composed of atoms occurring in facts listed in (11) *and* (12)*, and the universal closure of the following formulas:*

$$q(X) \wedge sum\big(set_{e1}(X)\big) \geq 1 \ \rightarrow\ p(X)$$
$$q(X) \wedge sum\big(set_{e2}(X)\big) \geq 1 \ \rightarrow\ s(X)$$
$$sum(set_{e3}) \geq 1 \ \rightarrow\ t$$

where $e1$ is the name for the set symbol $Y,Z : r(X,Y,Z)/X$; $e2$ is the name for $Y : r(X,Y,Z)/X$; and $e3$ is the name for $Y,Z : r(X,Y,Z)$. Note that the set symbols corresponding to names $e1$ and $e2$ have a global variable X. Consequently, function symbols set_{e1} and set_{e2} have arity $s_{prg} \rightarrow s_{set}$. The set symbol corresponding to $e3$ has no global variables. Consequently, set_{e3} is an object constant of sort s_{set}.

4.2 Semantics of Programs with Aggregates

For the sake of clarity, we describe the semantics of programs with aggregates in two steps. We start by assuming some restrictions on the form of interpretations of interest. These interpretations have fixed meanings for the symbols of signature σ_Π introduced in conditions c.-e. of its definition. In Section 6, we remove these restrictions on symbols *count*, *sum*, sum^+, *min*, *max*, and $set_{|E/\mathbf{X}|}$; and fix their meaning by providing appropriate axioms. In both cases, we assume the identity as the interpretation of the object constants and predicate constants stemming from the program's ground terms and predicate symbols, respectively. In other words, a program's ground terms satisfy the standard name assumption.

Let us consider some additional notation. 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 that contains variables from $\mathbf{X}$, $\alpha^{\mathbf{X}}_{\mathbf{x}}$ denotes the expression obtained from α by substituting $\mathbf{x}$ for $\mathbf{X}$. An *agg-interpretation I* is a many-sorted interpretation that satisfies the following *conditions*:

1. the domain $|I|^{s_{prg}}$ is the set containing all ground terms of program sort (or ground program terms, for short);

2. I interprets each ground program term as itself;

3. I interprets predicate symbols $>, \geq, <, \leq$ according to the total order chosen earlier;

4. universe $|I|^{s_{set}}$ is the set of all sets of non-empty tuples that can be formed with elements from $|I|^{s_{prg}}$;

5. if $E/\mathbf{X}$ is a set symbol, where E is an aggregate element of form (2), $\mathbf{Y}$ is the list of all variables occurring in E that are not in $\mathbf{X}$, and $\mathbf{x}$ and $\mathbf{y}$ are lists of ground program terms of the same length as $\mathbf{X}$ and $\mathbf{Y}$ respectively, then $set_{|E/\mathbf{X}|}(\mathbf{x})^I$ 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 $(l_1)^{\mathbf{XY}}_{\mathbf{xy}} \wedge \cdots \wedge (l_m)^{\mathbf{XY}}_{\mathbf{xy}}$;

6. for term t_{set} of sort s_{set}, $count(t_{set})^I$ is $\widehat{\texttt{count}}(t^I_{set})$;

7. for term t_{set} of sort s_{set}, $sum(t_{set})^I$ is $\widehat{\texttt{sum}}(t^I_{set})$;

8. for term t_{set} of sort s_{set}, $sum^+(t_{set})^I$ is $\widehat{\texttt{sum+}}(t^I_{set})$;

9. for term t_{set} of sort s_{set}, $min(t_{set})^I$ is $\widehat{\texttt{min}}(t^I_{set})$;

10. for term t_{set} of sort s_{set}, $max(t_{set})^I$ is $\widehat{\texttt{max}}(t^I_{set})$;

Recall that symbols $\widehat{\texttt{count}}$ through $\widehat{\texttt{max}}$ used in conditions 6 through 10 were defined in Section 3.1. Note how an agg-interpretation satisfies the standard name assumption for object constants of the program sort, but not for object constants and function constants of the set sort.

We say that an agg-interpretation I is a *stable model* of program Π if it satisfies the second-order sentence $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ with $\mathbf{p}$ being the list of all predicate symbols occurring in Π (note that this excludes predicate constants for the comparisons $>, \geq, <, \leq$).

In general, answer set solvers do not provide a complete first-order interpretation corresponding to a computed stable model. Rather, they list the set of ground atoms corresponding to it. Formally, for an agg-interpretation I, 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 stable model of Π, we say that $Ans(I)$ is an *answer set* of Π.

Example 3. *Take Π_1 to denote a program composed of rules (8-12). Let I be an agg-interpretation over σ_{Π_1} such that*

$$\begin{aligned} q^I &= \{a, b, c\} \\ r^I &= \{(a,1,a), (b,-1,a), (b,1,a), (b,1,b), (c,0,a)\}. \end{aligned} \tag{13}$$

Conditions 5 and 7 imply that this agg-interpretation also satisfies the following statements

$$\begin{array}{ll}
set_{e1}(a)^I = \{(1,a)\} & sum(set_{e1}(a))^I = 1 \\
set_{e1}(b)^I = \{(-1,a),(1,a),(1,b)\} & sum(set_{e1}(b))^I = 1 \\
set_{e1}(c)^I = \{(0,a)\} & sum(set_{e1}(c))^I = 0 \\
set_{e2}(a)^I = \{(1)\} & sum(set_{e2}(a))^I = 1 \\
set_{e2}(b)^I = \{(-1),(1)\} & sum(set_{e2}(b))^I = 0 \\
set_{e2}(c)^I = \{(0)\} & sum(set_{e2}(c))^I = 0 \\
set_{e3}^I = \{(1,a),(-1,a),(1,b),(0,a)\} & sum(set_{e3})^I = 1
\end{array} \tag{14}$$

Such an agg-interpretation I is a stable model of program Π_1 when $p^I = \{a,b\}$, $s^I = \{a\}$, and $t^I = \textbf{\textit{true}}$. It turns out, this program has a unique answer set

$$\begin{array}{l}
\{\, q(a), q(b), q(c), \\
\ r(a,1,a), r(b,-1,a), r(b,1,a), r(b,1,b), r(c,0,a), \\
\ p(a), p(b), s(a), t \,\}.
\end{array}$$

5. Relation with Abstract Gringo and ASP-Core-2 Semantics

In this section, we establish the correspondence between the semantics of programs with aggregates introduced in the previous section and the `Abstract Gringo` (Gebser et al., 2015) and `ASP-Core-2` (Calimeri et al., 2020) semantics. Both of these semantics can be stated in terms of infinitary formulas. The next subsection recalls the necessary definitions.

5.1 Background on Infinitary Formulas

We recall some definitions of infinitary logic (Truszczyński, 2012). 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 $\mathscr{H}$ is a set of formulas, and r is the smallest nonnegative integer that is greater than the ranks of all elements of $\mathscr{H}$, then $\mathscr{H}^\wedge$ and $\mathscr{H}^\vee$ are formulas of rank r (denoting the conjunction and disjunction of all formulas in set $\mathscr{H}$, respectively),
- 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 $\mathscr{A}$ and an infinitary formula is defined recursively:

- for every ground atom A from σ, $\mathscr{A} \models A$ if A belongs to $\mathscr{A}$,
- $\mathscr{A} \models \mathscr{H}^\wedge$ if for every formula F in $\mathscr{H}$, $\mathscr{A} \models F$,
- $\mathscr{A} \models \mathscr{H}^\vee$ if there is a formula F in $\mathscr{H}$ such that $\mathscr{A} \models F$,

- $\mathscr{A} \models F \to G$ if $\mathscr{A} \not\models F$ or $\mathscr{A} \models G$.

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

Stable Models The definitions of the `Abstract Gringo` and `ASP-Core-2` semantics rely on two different definitions of stable models, so-called *FT-stable* and *FLP-stable* models (Harrison & Lifschitz, 2019).

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

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

We say that a set $\mathscr{A}$ of ground atoms is an *FT-stable model* of an infinitary formula F if it is a $\subseteq$-minimal model of $F^{\mathscr{A}}$.

FLP-stable models are defined for sets of implications rather than arbitrary formulas. Let $\mathscr{H}$ 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* $FLP(\mathscr{H}, \mathscr{A})$ of $\mathscr{H}$ w.r.t. an interpretation $\mathscr{A}$ is the set of all formulas $G \to H$ from $\mathscr{H}$ such that $\mathscr{A}$ satisfies G. A set $\mathscr{A}$ of ground atoms is an *FLP-stable model* of $\mathscr{H}$ if it is a $\subseteq$-minimal model of $FLP(\mathscr{H}, \mathscr{A})$.

5.2 Relation with Abstract Gringo

We now establish the relation between the semantics of programs with aggregates proposed in Section 4.2 and the `Abstract Gringo` semantics (Gebser et al., 2015). This semantics captures the behavior of the answer set solver CLINGO when it evaluates a program with aggregates. Later, we derive the relation with the `ASP-Core-2` (Calimeri et al., 2020) semantics from the already-known link between the `Abstract Gringo` and `ASP-Core-2` semantics (Harrison & Lifschitz, 2019).

The `Abstract Gringo` semantics of logic programs uses a translation that turns a program into a set of infinitary propositional formulas (Gebser et al., 2015). The *FT-stable models* of the resulting set of infinitary propositional formulas define this semantics.

We now present a simplified version of the `Abstract Gringo` translation which is equivalent to the original in the studied fragment. A rule or an aggregate (in a rule) is called *closed* if it has no global variables. An *instance* of a rule R is any rule that can be obtained from R by substituting ground terms for all global variables.

For a closed aggregate element E of form (2) with $\mathbf{Y}$ being the list of non-global variables occurring in it, Ψ_E denotes the set of tuples $\mathbf{y}$ of ground program terms of the same length as $\mathbf{Y}$. Let E be an aggregate atom of form (3), Δ be a subset of Ψ_E and $[\Delta] = \{\mathbf{t}^{\mathbf{Y}}_{\mathbf{y}} \mid \mathbf{y} \in \Delta\}$ with $\mathbf{t}$ being the tuple $\langle t_1, \ldots, t_k \rangle$. Then, Δ *justifies* an aggregate atom if relation $\prec$ holds between $\widehat{\texttt{count}}([\Delta])$ $\big($resp. $\widehat{\texttt{sum}}([\Delta])$, $\widehat{\texttt{sum+}}([\Delta])$, $\widehat{\texttt{min}}([\Delta])$ and $\widehat{\texttt{max}}([\Delta])\big)$ and u. For example, let E_1 denote an aggregate element $3, X, Y : p(X,Y)$, then Ψ_{E_1} is the set of all tuples of ground program terms of length 2.

Let Δ_1 be a subset of Ψ_{E_1} composed of tuples $\{\langle a,b\rangle, \langle 5,b\rangle\}$. Then, $[\Delta_1] = \{\langle 3,a,b\rangle, \langle 3,5,b\rangle\}$. As a result, Δ_1 justifies a sample aggregate atom

$$\texttt{\#sum}\{3,X,Y : p(X,Y)\} \geq 5,$$

but Δ_1 does not justify another sample aggregate atom

$$\texttt{\#sum}\{3,X,Y : p(X,Y)\} \geq 7.$$

The `Abstract Gringo` translation τ is defined as follows:

1. for every ground atom A, its translation τA is A itself; $\tau\bot$ is $\bot$,
2. for every ground comparison $t_1 \prec t_2$, its translation $\tau(t_1 \prec t_2)$ is $\top$ if the relation $\prec$ holds between terms t_1 and t_2 according to the total order selected above and $\bot$ otherwise;
3. for aggregate atom A of form (3), τA is formula

$$\bigwedge_{\Delta\in\chi}\left(\bigwedge_{\mathbf{y}\in\Delta}\mathbf{l}_{\mathbf{y}}^{\mathbf{Y}} \rightarrow \bigvee_{\mathbf{y}\in\Psi_E\setminus\Delta}\mathbf{l}_{\mathbf{y}}^{\mathbf{Y}}\right) \tag{15}$$

   where χ is the set of subsets Δ of Ψ_E that do not justify A, and $\mathbf{l}$ stands for the conjunction $\tau l_1 \wedge \cdots \wedge \tau l_m$;
4. for every (basic or aggregate) literal L of form *not* A, its translation τL is $\neg\tau A$; if L is of form *not not* A, its translation τL is $\neg\neg\tau A$;
5. for every closed rule R of form (4), its translation τR is the implication

$$\tau B_1 \wedge \cdots \wedge \tau B_n \rightarrow \tau Head;$$

6. for every non-closed rule R, its translation τR is the conjunction of the result of applying τ to all its instances;
7. for every program Π, its translation $\tau\Pi$ is the infinitary theory containing τR for each rule R in Π.

A set $\mathscr{A}$ of ground atoms is a *gringo answer set* of a program Π if $\mathscr{A}$ is an FT-stable model of $\tau\Pi$.

Theorem 1. *The answer sets of any program (whose aggregates have no positive recursion) coincide with its gringo answer sets.*

5.3 Relation with ASP-Core-2 Semantics

Similarly to the `Abstract Gringo` semantics, the `ASP-Core-2` semantics relies on a translation of a program into an infinitary formula. Then, FLP-stable models of the resulting formula are used to characterize the `ASP-Core-2` semantics of the considered program. Originally, this semantics relied on a translation different from the one used in `Abstract Gringo`. Harrison and Lifschitz (2019) illustrated that in the context of FLP-stable models both translations are interchangeable. Note that the `ASP-Core-2` semantics is slightly more restrictive in the use of default negation – it

Predicate or function symbol	Arity
$\in$	$s_{tuple} \times s_{set}$
$+$	$s_{int} \times s_{int} \to s_{int}$
$\overline{\emptyset}$	s_{set}
$tuple_k$	$s_{prg} \times \cdots \times s_{prg} \to s_{tuple}$
rem	$s_{set} \times s_{tuple} \to s_{set}$
$first$	$s_{tuple} \to s_{prg}$
$weight$	$s_{tuple} \to s_{int}$

Table 3: Predicate and function symbols for representing aggregate atoms. We assume that $k \geq 1$.

does not allow double negated literals; and it does not allow the use of operation `sum+`. Also, as mentioned earlier, the `ASP-Core-2` semantics is introduced for programs that have no recursion over aggregates. We refer to programs satisfying these stated restrictions as *core programs*. In this context, we can say that a set $\mathscr{A}$ of ground atoms is a *core answer set* of a core program Π if $\mathscr{A}$ is an FLP-stable model of $\tau\Pi$. That is, the essential difference between gringo and core answer sets is their reliance on FT-stable and FLP-stable models, respectively. The Main Theorem by Harrison and Lifschitz (2019) states that for a class of programs that includes core programs the FLP-stable models coincide with the FT-stable models.

A direct consequence of the Main Theorem in (Harrison & Lifschitz, 2019) is the fact that the gringo answer sets of any core program coincide with its core answer sets. Consequently, by Theorem 1 we derive that for core programs all three semantics discussed in the paper coincide.

Theorem 2. *The answer sets of any core program coincide with its core answer sets (and gringo answer sets).*

6. Axiomatization of Aggregates

In this section we show that within the definition of agg-interpretations conditions 5-10 can be removed from the meta-logic level by adding new logical sentences to the theory representing a logic program. This provides higher mathematical rigor and allows us to build object-level proofs to reason about programs with aggregates.

We introduce an extended signature σ^*_Π that expands σ_Π with new symbols and new sorts. The new sorts are s_{int} and s_{tuple} that we refer to as *integer* and *tuple*, respectively. Table 3 lists the new symbols and their associated sorts. We also assume countably infinite sets of integer and tuple variables (variables of sorts s_{int} and s_{tuple}). We use the letter N and its variants to denote integer variables and the letter T and its variants to denote tuple variables. Letters V, W and their variants denote variables, where the sort is explicitly mentioned.

As customary in mathematics, we use infix notation for the function symbol $+$ and the predicate symbol $\in$. Informally, $tuple_k(t_1,\dots,t_k)$ is a constructor for the k-tuple containing program terms $t_1,\dots,t_k$; atomic formula $t_{tuple} \in t_{set}$ holds if and only if tuple t_{tuple} belongs to set t_{set}; $rem(t_{set},t_{tuple})$ encodes the set obtained by removing tuple t_{tuple} from set t_{set}; $first(t_{tuple})$ is the first element of tuple t_{tuple}; and $weight(t_{tuple})$ encodes the weight of tuple t_{tuple} (recall that the syntactic object t_{tuple} is meant to be interpreted as an object of sort s_{tuple}). Note that when the first element c of a tuple t_{tuple} is a symbolic constant, $weight(t_{tuple})$ is 0 but $first(t_{tuple})$ is c. This choice could be made differently, but we follow this convention to capture the `Abstract Gringo` and `ASP-Core-2` semantics.

For signature σ^*_Π we extend the set of *conditions* that an *agg-interpretation I* satisfies:

11. the domain $|I|^{s_{int}}$ is the set of all numerals;
12. I interprets $\overline{m}+\overline{n}$ as $\overline{m+n}$,
13. universe $|I|^{s_{tuple}}$ is the set of all tuples of form $\langle d_1,\ldots,d_m\rangle$ with $m \geq 1$ and each $d_i \in |I|^{s_{prg}}$;
14. I interprets each tuple term of form $tuple_k(t_1,\ldots,t_k)$ as tuple $\langle t_1^I,\ldots,t_k^I\rangle$;
15. I interprets object constant $\overline{\emptyset}$ as the empty set $\emptyset$;
16. I satisfies $t_1 \in t_2$ if and only if tuple t_1^I belongs to set t_2^I;
17. $rem(t_{set},t_{tuple})^I$ is the set obtained by removing tuple t^I_{tuple} from set t^I_{set};
18. $first(t_{tuple})^I$ is the first element of t^I_{tuple}.
19. $weight(t_{tuple})^I$ is the weight of t^I_{tuple}.

Note that $|I|^{s_{set}}$ is the power set of $|I|^{s_{tuple}}$. Also, each agg-interpretation is extended in a unique way: there is a one-to-one correspondence between the agg-interpretations over σ_Π and σ^*_Π. In the sequel, we identify each agg-interpretation in signature σ_Π with its extension in σ^*_Π.

In the remainder of this section, we show how an agg-interpretation can be "axiomatized" in a theory that interprets symbols for arithmetic, tuples, sets, and program object constants in a standard way. Formally, a first-order interpretation I is called *standard* when it satisfies conditions 1-4 and 11-16. Such an interpretation satisfies the standard name assumption for ground program terms and tuples, the standard interpretation of arithmetic symbols, and the standard interpretation of the set-theoretic membership predicate. It does not assign any special meaning to symbols *count*, *sum*, sum^+, *min*, *max*, *rem*, *first*, *weight*, or any of the functions constants of the form $set_{|E/\mathbf{X}|}$. It is obvious that every agg-interpretation is also a standard interpretation, but not vice-versa.

In the following subsections, we show that agg-interpretations can be characterized as standard interpretations that satisfy a particular class of sentences. The subsections cover various conditions of agg-interpretations. Theorem 3 states a key theoretical result that binds the stated findings together. The Main Theorem closes this section by relating this key result to core programs.

6.1 Axiomatizing Set Symbols (Condition 5)

Let us start by considering condition 5 of the agg-interpretation definition. It associates a set symbol $E/\mathbf{X}$, where E has the form (2), with a unique set. We characterize this set by the sentence

$$\forall \mathbf{X} T\big(T \in set_{|E/\mathbf{X}|}(\mathbf{X}) \leftrightarrow \exists \mathbf{Y}(T = tuple_k(t_1,\ldots,t_k) \wedge l_1 \wedge \cdots \wedge l_m)\big) \tag{16}$$

where $\mathbf{Y}$ is the list of all the variables occurring in E that are not in $\mathbf{X}$.

Example 4. *Recall program* Π_1 *introduced in Section 4.2 and the aggregate symbol* $Y,Z : r(X,Y,Z)/X$ *named* $e1$. *For symbol* $e1$*, sentence* (16) *has the form*

$$\forall XT\Big(T \in set_{e1}(X) \leftrightarrow \exists YZ\left(T = tuple_2(Y,Z) \wedge r(X,Y,Z)\right)\Big)$$

For a sample standard interpretation I over signature $\sigma^*_{\Pi_1}$ satisfying conditions in (13) *and this sentence, $set_{e1}(b)^I$ is set $\{(-1,a),(1,a),(1,b)\}$. This set is identical to the one listed in* (14) *for an agg-interpretation satisfying* (13)*. For aggregate symbol $Y,Z : r(X,Y,Z)$ named $e3$, sentence* (16) *has the form*

$$\forall T\,(T \in set_{e3} \leftrightarrow \exists XYZ\,(T = tuple_2(Y,Z) \wedge r(X,Y,Z)))$$

and, for the same interpretation, we can see that set_{e3} is the set containing tuples corresponding to $tuple_2(Y,Z)$ such that I satisfies $r(X,Y,Z)$ for some X. Consequently, set^I_{e3} is set

$$\{(1,a),(-1,a),(1,b),(0,a)\}.$$

Once more this set is identical to the one listed in (14)*.*

The conclusions of Example 4 hint at a general result stated next.

Proposition 1. *Let I be a standard interpretation. Then, I satisfies condition 5 if and only if it satisfies sentence* (16) *for every function symbol of form $set_{|E/\mathbf{X}|}$.*

As another example, consider the second rule in Listing 1. The result of applying the translation κ to this rule is the universal closure of the following first-order formula:

$$vertex(X) \wedge \neg count(set_{e_4}(X)) = 1 \rightarrow \bot$$

where e_4 is the name associated with set symbol $X,Z : assign(X,Z), color(Z)/X$. For symbol e_4, sentence (16) has the form

$$\forall XT\,(T \in set_{e_4}(X) \leftrightarrow \exists Z\,(T = tuple_2(X,Z) \wedge assign(X,Z) \wedge color(Z)))\,.$$

6.2 Axiomatizing of Tuple Operations (Conditions 17, 18, and 19)

The meaning of function symbols *rem*, *first*, and *weight* provided by conditions 17, 18, and 19 of the definition of agg-interpretations can be fixed in standard interpretations using the following sentences:

$$\forall STS'\big(rem(S,T) = S' \leftrightarrow \forall T'\big(T' \in S' \leftrightarrow (T' \in S \wedge T' \neq T)\big)\big) \tag{17}$$

$$\forall X_1 \ldots X_k\,\big(first(tuple_k(X_1,\ldots,X_k)) = X_1\big) \tag{18}$$

$$\forall NX_2 \ldots X_k\,\big(weight(tuple_k(N,X_2,\ldots,X_k)) = N\big) \tag{19}$$

$$\forall X_1 \ldots X_k((\neg\exists N\,X_1 = N) \rightarrow weight(tuple_k(X_1,X_2,\ldots,X_k)) = 0) \tag{20}$$

Sentences (18-20) are axiom schemata and should be included for every function symbol $tuple_k$ occurring in the program.

Proposition 2. *Let I be a standard interpretation. Then,*

- *I satisfies condition 17 if and only if it satisfies sentence* (17)*;*
- *I satisfies condition 18 if and only if it satisfies all sentences of form* (18)*; and*
- *I satisfies condition 19 if and only if it satisfies all sentences of form (19-20).*

6.3 Axiomatizing *count* (Condition 6)

Formalizing condition 6 requires determining when a set is finite or not, that is, we need a formula $\mathit{Finite}(t_{set})$ that holds if and only if the set represented by t_{set} is finite. We can formalize this idea using a second-order formula, which states that *there is a natural number N and an injective function from t_{set} into the set* $\{i \in \mathbb{N} \mid i \leq N\}$. Before formalizing this statement, let us introduce some auxiliary definitions. Given a term t_{set} of sort s_{set} and a function symbol f, we define $\mathit{Injective}(f, t_{set})$ as the formula

$$\forall T_1 T_2 \, (T_1 \in t_{set} \wedge T_2 \in t_{set} \wedge f(T_1) = f(T_2) \rightarrow T_1 = T_2.)$$

Intuitively, formula $\mathit{Injective}(f, t_{set})$ represents the fact that the restriction of function f to the set corresponding to the elements of the set corresponding to term t_{set} is injective. If the image of f is of sort s_{prg} and t_1 and t_2 are also terms of sort s_{prg}, we define $\mathit{Image}(f, t_{set}, t_1, t_2)$ as the formula:

$$\forall T \, (T \in t_{set} \rightarrow t_1 \leq f(T) \wedge f(T) \leq t_2)$$

Formula $\mathit{Image}(f, t_{set}, t_1, t_2)$ holds when the image of the restriction of function f to the set corresponding to t_{set} is between the values represented by terms t_1 and t_2. Expression $\mathit{Finite}(t_{set})$ stands for the second-order formula

$$\exists f \, (\mathit{Injective}(f, t_{set}) \wedge \exists N \; \mathit{Image}(f, t_{set}, 0, N))$$

where f is a function variable of arity $s_{tuple} \rightarrow s_{int}$. Intuitively, this expression states the already mentioned statement: *there is a natural number N and an injective function from t_{set} into the set* $\{i \in \mathbb{N} \mid i \leq N\}$.

For a term t_{set} of the set sort, we define formula $\mathit{FiniteCount}(t_{set})$ as

$$\forall T \left(T \in t_{set} \rightarrow \exists N \left(\mathit{count}(\mathit{rem}(t_{set}, T)) = N \wedge \mathit{count}(t_{set}) = N + \overline{1} \right) \right)$$

Intuitively, this sentence states that the number of elements of any non-empty finite set is the result of adding one to any set obtained from it by removing a single element.

Using these formulas we can formalize condition 6 with the help of the following three sentences:

$$\mathit{count}(\overline{\emptyset}) = \overline{0} \tag{21}$$

$$\forall S \, (\mathit{Finite}(S) \rightarrow \mathit{FiniteCount}(S)) \tag{22}$$

$$\forall S \, (\neg \mathit{Finite}(S) \rightarrow \mathit{count}(S) = \mathit{sup}) \tag{23}$$

Proposition 3. *Let I be an interpretation that satisfies all conditions for being an agg-interpretation except conditions 6-10. Then, I satisfies condition 6 if and only if it satisfies sentences (21-23).*

Note that the statements of Propositions 1 and 2 concern standard interpretations. The statement of Proposition 3 (and propositions in the following subsections) concerns interpretations that satisfy all conditions for being an agg-interpretation except conditions 6-10. Such interpretations differ from standard ones *only* by a requirement that they have to satisfy condition 5 of the agg-interpretation definition. Alternatively, due to Proposition 1 such interpretations satisfy sentence (16).

6.4 Axiomatizing *sum* (Condition 7)

The axiomatization of aggregates with the operation *sum* is similar to the case of *count*, but requires characterizing that the set of tuples with non-zero weight is finite (instead of the set of arbitrary tuples). Given a term t_{set} of sort s_{set} and a function symbol f, we define $\mathit{InjectiveWeight}(f, t_{set})$ as the formula

$$\forall T_1 T_2 \big(T_1 \in t_{set} \wedge T_2 \in t_{set} \wedge \mathit{weight}(T_1) \neq 0 \wedge \mathit{weight}(T_2) \neq 0 \wedge f(T_1) = f(T_2) \rightarrow T_1 = T_2\big).$$

Similar to $\mathit{Injective}(f, t_{set})$, this formula represents the fact that the restriction of function f to the set of elements of t_{set} with non-zero weight is injective.

If the image of f is of sort s_{prg} and t_1 and t_2 are also terms of sort s_{prg}, we define $\mathit{ImageWeight}(f, t_{set}, t_1, t_2)$ as formula

$$\forall T \big(T \in t_{set} \wedge \mathit{weight}(T) \neq 0 \rightarrow t_1 \leq f(T) \wedge f(T) \leq t_2\big).$$

Similar to $\mathit{Image}(f, t_{set}, t_1, t_2)$, this formula represents the fact that the image of the restriction of function f to the set of elements of t_{set} with non-zero weight is an integer between the integers represented by terms t_1 and t_2. Expression $\mathit{FiniteWeight}(t_{set})$ stands for the second-order formula

$$\exists f \big(\mathit{InjectiveWeight}(f, t_{set}) \wedge \exists N\ \mathit{ImageWeight}(f, t_{set}, 0, N)\big)$$

where f is a function variable of arity $s_{tuple} \rightarrow s_{int}$. For a term t_{set} of the set sort, we define formula $\mathit{FiniteSum}(t_{set})$ as

$$\forall T \big(T \in t_{set} \rightarrow \exists N(\mathit{sum}(\mathit{rem}(t_{set}, T)) = N \wedge \mathit{sum}(t_{set}) = N + \mathit{weight}(T))\big)$$

Intuitively, this sentence states that the sum of elements of any finite non-empty set is the result of adding the weight of any of its elements to the sum of elements of the set obtained from it by removing that element.

We can define *sum* to have arity $s_{set} \rightarrow s_{int}$ and simplify the formula that stands for $\mathit{FiniteSum}(t_{set})$ as follows:

$$\forall T \big(T \in t_{set} \rightarrow \mathit{sum}(t_{set}) = \mathit{sum}(\mathit{rem}(t_{set}, T)) + \mathit{weight}(T)\big)$$

Note that a similar simplification cannot be made for *count*, for example, because sometimes it returns *sup*, which is not of sort int. We also define $\mathit{ZeroWeight}(t_{set})$ as

$$\forall T\, (T \in t_{set} \rightarrow \mathit{weight}(T) = 0)$$

which holds when all members of t_{set} have zero-weight.

Using these formulas we can formalize condition 7 with the help of the following three sentences:

$$\forall S \big(\mathit{ZeroWeight}(S) \rightarrow \mathit{sum}(S) = \overline{0}\big) \tag{24}$$

$$\forall S\, (\mathit{FiniteWeight}(S) \rightarrow \mathit{FiniteSum}(S)) \tag{25}$$

$$\forall S \big(\neg \mathit{FiniteWeight}(S) \rightarrow \mathit{sum}(S) = \overline{0}\big) \tag{26}$$

In particular, note that (24) entails $\mathit{sum}(\overline{\emptyset}) = \overline{0}$.

Proposition 4. *Let I be an interpretation that satisfies all conditions for being an agg-interpretation except conditions 6-10. Then, I satisfies condition 7 if and only if it satisfies sentences (24-26).*

6.5 Axiomatizing *sum+* (Condition 8)

Axiomatizing *sum+* is similar to the case of *sum* captured in previous subsection. We adapt to the condition where only positive weights are to be considered. We define formula $\mathit{FinitePositive}(t_{set})$ which holds when t_{set} contains finitely many tuples with positive weight. We construct this formula in a similar way to $\mathit{FiniteWeight}(S)$, by first defining *InjectivePositive* and *ImagePositive*. For a term t_{set} of sort s_{set} and a function symbol f, let $\mathit{InjectivePositive}(f,t_{set})$ denote the formula

$$\forall T_1 T_2 \big(T_1 \in t_{set} \wedge T_2 \in t_{set} \wedge \mathit{weight}(T_1) > 0 \wedge \\ \mathit{weight}(T_2) > 0 \wedge f(T_1) = f(T_2) \rightarrow T_1 = T_2\big)$$

When the image of f is of sort s_{prg} and t_1 and t_2 are also terms of sort s_{prg}, then $\mathit{ImagePositive}(f,t_{set},t_1,t_2)$ denotes the formula

$$\forall T \big(T \in t_{set} \wedge \mathit{weight}(T) > 0 \rightarrow t_1 \leq f(T) \wedge f(T) \leq t_2\big).$$

Expression $\mathit{FinitePositive}(t_{set})$ stands for the second-order formula

$$\exists f \big(\mathit{InjectivePositive}(f,t_{set}) \wedge \exists N\ \mathit{ImagePositive}(f,t_{set},0,N)\big)$$

where f is a function variable of arity $s_{tuple} \rightarrow s_{int}$. Finally, we must account for the case when none of the tuples in our set t_{set} have positive weight with the formula $\mathit{NonPositiveWeight}(t_{set})$:

$$\forall T\ (T \in t_{set} \rightarrow \mathit{weight}(T) \leq 0)$$

Let $\mathit{FinitePositiveSum}(t_{set})$ be the formula

$$\forall T \big(T \in t_{set} \wedge \mathit{weight}(T) > 0 \rightarrow \exists N(\mathit{sum}^+(\mathit{rem}(t_{set},T)) = N \wedge \\ \mathit{sum}^+(t_{set}) = N + \mathit{weight}(T))\big)$$

Now we formalize condition 8 as follows:

$$\forall S \big(\mathit{NonPositiveWeight}(S) \rightarrow \mathit{sum}^+(S) = \overline{0}\big) \tag{27}$$

$$\forall S\,(\mathit{FinitePositive}(S) \rightarrow \mathit{FinitePositiveSum}(S)) \tag{28}$$

$$\forall S \big(\neg \mathit{FinitePositive}(S) \rightarrow \mathit{sum}^+(S) = \mathit{sup}\big) \tag{29}$$

Proposition 5. *Let I be an interpretation that satisfies all conditions for being an agg-interpretation except conditions 6-10. Then, I satisfies condition 8 if and only if it satisfies sentences (27-29).*

6.6 Axiomatizing *min* and *max* (Conditions 9 and 10)

The axiomatization of *min* and *max* are simpler than previous cases because we do not need to define their value recursively. We define $\mathit{FiniteMin}(t_{set})$ and $\mathit{FiniteMax}(t_{set})$, respectively, as follows

$$\forall T \big(T \in t_{set} \wedge \neg \exists T' \big(T' \in t_{set} \wedge \mathit{first}(T') < \mathit{first}(T)\big) \rightarrow \mathit{min}(t_{set}) = \mathit{first}(T)\big)$$
$$\forall T \big(T \in t_{set} \wedge \neg \exists T' \big(T' \in t_{set} \wedge \mathit{first}(T') > \mathit{first}(T)\big) \rightarrow \mathit{max}(t_{set}) = \mathit{first}(T)\big).$$

The following sentences formalize condition 9:

$$\mathit{min}(\overline{\emptyset}) = \mathit{sup} \tag{30}$$

$$\forall S\,(\mathit{Finite}(S) \rightarrow \mathit{FiniteMin}(S)) \tag{31}$$

$$\forall S\,(\neg \mathit{Finite}(S) \rightarrow \mathit{min}(S) = \mathit{inf}) \tag{32}$$

Proposition 6. *Let I be an interpretation that satisfies all conditions for being an agg-interpretation except conditions 6-10. Then, I satisfies condition 9 if and only if it satisfies sentences (30-32).*

Similarly, condition 10 is formalized using the following sentences:

$$max(\overline{\emptyset}) = inf \tag{33}$$
$$\forall S\,(Finite(S) \to FiniteMax(S)) \tag{34}$$
$$\forall S\,(\neg Finite(S) \to max(S) = sup) \tag{35}$$

Proposition 7. *Let I be an interpretation that satisfies all conditions for being an agg-interpretation except conditions 6-10. Then, I satisfies condition 10 if and only if it satisfies sentences (33-35).*

The theorem below follows directly from Propositions 1-7.

Theorem 3. *A set M of ground atoms is an answer set of a program* Π *if and only if there exists some standard model I of* $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ *that satisfies all sentences of form (16-35) and* $M = Ans(I)$*. Symbol* $\mathbf{p}$ *refers to the list of all predicate symbols occurring in* Π*.*

6.7 Main Theorem

Combining Theorems 2 and 3, we obtain our main result: the axiomatization of aggregates according to the `ASP-Core-2` semantics.

Theorem 4. *A set M of ground atoms is a core answer set of a core program* Π *if and only if there exists some standard model I of* $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ *that satisfies all sentences of form (16-35) and* $M = Ans(I)$*. Symbol* $\mathbf{p}$ *refers to the list of all predicate symbols occurring in* Π*.*

If we relax the syntactic conditions of the `ASP-Core-2` and we allow double negated literals and non-positive recursive aggregates, we obtain that our axiomatization coincides with `Abstract Gringo` semantics. The following result parallels the Main Theorem, and follows directly from Theorems 1 and 3.

Theorem 5. *A set M of ground atoms is a gringo answer set of a program* Π *without positive recursive aggregates if and only if there exists some standard model I of* $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ *that satisfies all sentences of form (16-35) and* $M = Ans(I)$*. Symbol* $\mathbf{p}$ *refers to the list of all predicate symbols occurring in* Π*.*

7. First-Order Characterization

There is a wide class of programs without aggregates for which the second-order SM operator can be replaced by a first-order formalization. This includes completion in the case of tight programs (Ferraris et al., 2011) or, more generally, loop formulas (Lee & Meng, 2011) and ordered completion for finite structures (Asuncion, Lin, Zhang, & Zhou, 2012). In particular, the Completion Theorem by Ferraris et al. (2011, Theorem 11) only applies to one-sorted formulas in the so-called *Clark normal* form that are tight. Recently, a more general notion of "locally tight" was introduced by Fandinno and Lifschitz (2023). In that work, the authors generalized completion to the case of many-sorted formulas that go beyond Clark normal form. The Main Lemma of the paper stated that when the positive dependency graph of these formulas do not have infinite walks, then the models of completion coincide with the stable models (as understood here) of the formula. There is a price to

pay for using the notion of locally tight, as the task of verifying whether a formula is locally tight is, in general, undecidable. Here we define tightness for the kind of formulas considered by Fandinno and Lifschitz (2023). This allows us to state a formal result for the equivalence of completion and programs with aggregates when they are tight.

7.1 Completion and Tightness

For a set $\mathbf{p}$ of predicate symbols, a *nondisjunctive implication* is a formula that has the form

$$\tilde{\forall}(F \to G), \tag{36}$$

where G is an atomic formula or a formula that does not contain predicate symbols in $\mathbf{p}$. Given this implication, we say that it *defines* a predicate symbol p if G is an atomic formula with predicate symbol p and p belongs to $\mathbf{p}$. We say that this implication is a *constraint* if G does not contain predicate symbols in $\mathbf{p}$.

Let Γ be a finite set of nondisjunctive implications. If the argument sorts of a predicate symbol p in $\mathbf{p}$ are $s_1 \ldots, s_n$, and the members of Γ defining p are

$$\tilde{\forall}(F_i \to p(\mathbf{t}_i)) \qquad i = 1, \ldots, k,$$

then the *completed definition* of p in Γ is the sentence

$$\forall \mathbf{V} \left(p(\mathbf{V}) \leftrightarrow \bigvee_{i=1}^{k} \exists \mathbf{U}_i \, (F_i \wedge \mathbf{V} = \mathbf{t}_i) \right), \tag{37}$$

where $\mathbf{V}$ is an n-tuple of fresh variables of sorts $s_1, \ldots, s_n$, and $\mathbf{U}_i$ is the list of all variables that are free in $F_i \to p(\mathbf{t}_i)$. The expression $\mathbf{V} = \mathbf{t}_i$ stands here for the conjunction of n equalities between the corresponding members of the tuples $\mathbf{V}$ and $\mathbf{t}_i$.

The *completion* $\mathrm{COMP}_{\mathbf{p}}[\Gamma]$ of Γ is the conjunction of the completed definitions in Γ of all predicate symbols p in $\mathbf{p}$ and all constraints of Γ.

Example 5. *If* $\mathbf{p}$ *is the list containing only the predicate symbol* $assign/1$*, then the completion of the program in Listing 1 is equivalent in first-order logic to the following sentences:*

$$\begin{aligned} \forall X\, Z(assign(X,Z) &\leftrightarrow vertex(X) \wedge color(Z) \wedge \neg\neg assign(X,Z)) \\ \forall X(vertex(X) &\wedge \neg count(set_{asg}(X)) = 1 \to \bot) \\ \forall X\, Y\, Z(edge(Y,X) &\wedge assign(Y,Z) \wedge assign(X,Z) \to \bot) \end{aligned}$$

where asg is the name for the set symbol $X, Y : assign(X,Y), color(Y)/X$.

The following result is an immediate consequence of Lemma 4 by Fandinno and Lifschitz (2023).

Theorem 6. *For any finite set* Γ *of nondisjunctive implications and list* $\mathbf{p}$ *of predicate symbols, the implication*

$$\mathrm{SM}_{\mathbf{p}}[\Gamma] \to \mathrm{COMP}_{\mathbf{p}}[\Gamma]$$

is logically valid.

Combining this Theorem with our Main Theorem and Theorem 5, we obtain the following results for core and gringo answer sets.

Corollary 1. *For a core program* Π*, if a set M of ground atoms is a core answer set of* Π*, then there exists some standard model I of* $\mathrm{COMP}_{\mathbf{p}}[\kappa\Pi]$ *that satisfies all sentences of form (16-35) and* $M = Ans(I)$*. Symbol* $\mathbf{p}$ *refers to the list of all predicate symbols occurring in* Π*.*

Corollary 2. *For a program* Π *(without positive recursive aggregates), if a set M of ground atoms is a gringo answer set of* Π*, then there exists some standard model I of* $\mathrm{COMP}_{\mathbf{p}}[\kappa\Pi]$ *that satisfies all sentences of form (16-35) and* $M = Ans(I)$*. Symbol* $\mathbf{p}$ *refers to the list of all predicate symbols occurring in* Π*.*

The converse of Theorem 6 is not true in general, but it holds for the class of *tight* theories.

For any finite set Γ of nondisjunctive implications, the *predicate dependency graph* of Γ with respect to a list $\mathbf{p}$ of predicate symbols is the directed graph that

- has all predicates of $\mathbf{p}$ as its vertices, and
- has an edge from p to q if there is a formula of the form of (36) in Γ with p in the consequent and a strictly positive occurrence of q in the antecedent.

We say that Γ is *tight* with respect to $\mathbf{p}$ if the predicate dependency graph of Γ with respect to $\mathbf{p}$ is acyclic.

Theorem 7. *For any tight finite set* Γ *of nondisjunctive implications and list of predicate symbols* $\mathbf{p}$*, the equivalence*

$$\mathrm{SM}_{\mathbf{p}}[\Gamma] \leftrightarrow \mathrm{COMP}_{\mathbf{p}}[\Gamma]$$

is logically valid.

Theorem 7 is a consequence of the Main Lemma by Fandinno and Lifschitz (2023). The proof of this result relies on the observation that, for any tight program, its infinite dependency graph does not have infinite walks. Combining this theorem with our Main Theorem and Theorem 5, we obtain the following characterizations for the `ASP-Core-2` and `Abstract Gringo` semantics in terms of completion.

Corollary 3. *Let* Π *be a core program such that* $\kappa\Pi$ *is tight. Then, a set M of ground atoms is a core answer set of* Π *if and only if there exists some standard model I of* $\mathrm{COMP}_{\mathbf{p}}[\kappa\Pi]$ *that satisfies all sentences of form (16-35) and* $M = Ans(I)$*. Symbol* $\mathbf{p}$ *refers to the list of all predicate symbols occurring in* Π*.*

Corollary 4. *Let* Π *be a program (without positive recursive aggregates) such that* $\kappa\Pi$ *is tight. Then, a set M of ground atoms is a gringo answer set of* Π *if and only if there exists some standard model I of* $\mathrm{COMP}_{\mathbf{p}}[\kappa\Pi]$ *that satisfies all sentences of form (16-35) and* $M = Ans(I)$*. Symbol* $\mathbf{p}$ *refers to the list of all predicate symbols occurring in* Π*.*

7.2 First-Order Axioms for Aggregates

Even when we can replace the SM operator by the completion operator as done in Corollaries 1- 4, our translation still relies on second-order formulas because some axioms added to characterize agg-interpretations by means of standard interpretations are second-order formulas. To see this, consider the quantification over function symbols in the formulas $Finite(t_{set})$, $FiniteWeight(t_{set})$ and $FinitePositive(t_{set})$. These formulas are necessary to distinguish between finite and infinite

Second-order Sentences	First-order Replacements
(22-23)	(38)
(25-26)	(39)
(28-29)	(40)
(31-32)	(41)
(34-35)	(42)

Table 4: First-order characterizations.

sets. In practice, the standard `ASP-Core-2` states that "to promote declarative programming as well as practical system implementation, ASP-Core-2 programs are supposed to comply with the restrictions." These restrictions include, among others, the requirement that programs have finite answer sets and result only in finite sets of aggregate elements (Calimeri et al., 2020, Sections 2 and 5).

Formally, we say that an interpretation I *has finite aggregates* if sets of the form $set_{|E/\mathbf{X}|}(\mathbf{x})^I$ are finite for every set symbol $E/\mathbf{X}$ and any list $\mathbf{x}$ of ground program terms of the same length as $\mathbf{X}$. A program Π *has finite aggregates* if all standard models of $\mathrm{SM}[\kappa\Pi]$ have finite aggregates.

In the rest of this section, we focus on programs with finite aggregates and we disregard how this property is obtained. Under these conditions, we are able to provide an axiomatization that bypasses the need for second-order formulas. Consequently, in the case of tight programs, the result on completion and this new axiomatization paves the way to capture the semantics of programs with aggregates by means of first-order logic. As mentioned earlier, this is an interesting fragment from a practical point of view.

Given two terms t_{set}, t'_{set} of the set sort, we define the formula $Subset(t_{set}, t'_{set})$ as

$$\forall T \left(T \in t_{set} \rightarrow T \in t'_{set} \right)$$

stating that t_{set} is a subset of t'_{set}. In the case of programs that have finite aggregates, we can replace our second-order sentences with the following first-order ones, where $E/\mathbf{X}$ is an aggregate symbol (see Table 4):

$$\forall \mathbf{X}\, S \left(Subset(S, set_{|E/\mathbf{X}|}(\mathbf{X})) \rightarrow FiniteCount(S) \right) \tag{38}$$

$$\forall \mathbf{X}\, S \left(Subset(S, set_{|E/\mathbf{X}|}(\mathbf{X})) \rightarrow FiniteSum(S) \right) \tag{39}$$

$$\forall \mathbf{X}\, S \left(Subset(S, set_{|E/\mathbf{X}|}(\mathbf{X})) \rightarrow FinitePositiveSum(S) \right) \tag{40}$$

$$FiniteMin(set_{|E/\mathbf{X}|}(\mathbf{X})) \tag{41}$$

$$FiniteMax(set_{|E/\mathbf{X}|}(\mathbf{X})) \tag{42}$$

Intuitively, each pair of second-order sentences has the same meaning as their first-order replacement, but with some restrictions. First, this formalization is appropriate only if the interpretation of $set_{|E/\mathbf{X}|}(\mathbf{x})$ results in a finite set. Furthermore, the interpretation of *count*, *sum* and sum^+ is only fixed for subsets of sets corresponding to terms of the form $set_{|E/\mathbf{X}|}(\mathbf{x})$. Hence, there may be non-standard interpretations that satisfy these sentences. These non-standard interpretations may assign values that do not correspond to their intended meaning to some sets, but we can guarantee that they assign the intended meaning to all symbols of the sort set that occur in the program translation. The reason to include all subsets in these sentences is that $FiniteCount(S)$, $FiniteSum(S)$,

Listing 3: An instance of the graph coloring problem containing three colors and a graph with three vertices and three edges.

```
color(r).      color(g).      color(b).
vertex(1).     vertex(2).     vertex(3).
edge(1,2).     edge(2,3).     edge(3,1).
```

and *FinitePositiveSum*(S) recursively refer to some of their subsets. Since *FiniteMin* and *FiniteMax* do not refer to their subsets, we don't need to include all subsets. The following result shows that in the case of programs with finite aggregates we can use the introduced first-order axiomatization.

Theorem 8. *A set M of ground atoms is an answer set of some program Π with finite aggregates iff there exists some standard model I of $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ that satisfies all sentences of form (16-21,24,27,30,33) and all sentences of form (38-42), and $M = Ans(I)$.*

Combining this theorem with our Main Theorem, Theorem 5, and Theorem 7 we obtain the following characterizations for the `ASP-Core-2` and `Abstract Gringo` semantics in terms of first-order formulas.

Corollary 5. *Let Π be a core program with finite aggregates such that $\kappa\Pi$ is tight. A set M of ground atoms is a core answer set of Π if and only if there exists some standard model I of $\mathrm{COMP}_{\mathbf{p}}[\kappa\Pi]$ that satisfies all sentences of form (16-21,24,27,30,33) and all sentences of form (38-42), and $M = Ans(I)$. Symbol $\mathbf{p}$ refers to the list of all predicate symbols occurring in Π.*

Corollary 6. *Let Π be a program (without positive recursive aggregates) with finite aggregates such that $\kappa\Pi$ is tight. A set M of ground atoms is a gringo answer set of Π if and only if there exists some standard model I of $\mathrm{COMP}_{\mathbf{p}}[\kappa\Pi]$ that satisfies all sentences of form (16-21,24,27,30,33) and all sentences of form (38-42), and $M = Ans(I)$. Symbol $\mathbf{p}$ refers to the list of all predicate symbols occurring in Π.*

Example 6. *Let Π be the program obtained by adding facts in Listing 3 to the program in Listing 1 and let $\mathbf{p}$ be the list of all predicate symbols in Π. Then, the completion of Π, in symbols $\mathrm{COMP}_{\mathbf{p}}[\kappa\Pi]$, is the result of adding the following formulas to the set of formulas in Example 5:*

$$\forall X\,(color(X) \leftrightarrow (X = r \vee X = g \vee X = b))$$
$$\forall X\,(vertex(X) \leftrightarrow (X = 1 \vee X = 2 \vee X = 3))$$
$$\forall X\,Y\,(edge(X,Y) \leftrightarrow (X = 1 \wedge Y = 2 \vee X = 2 \wedge Y = 3 \vee X = 3 \wedge Y = 1))$$

Let Δ denote the set of axioms of form (16-21), (24)*,* (27)*,* (30)*,* (33) *and (38-42). Since $\kappa\Pi$ is tight, Corollary 6 guarantees that we can compute the gringo answer sets of Π by computing the standard models of the first-order theory $\mathrm{COMP}_{\mathbf{p}}[\kappa\Pi] \cup \Delta$. Let I be a standard interpretation such that*

$$edge^I = \{(1,2),(2,3),(3,1)\} \qquad vertex^I = \{1,2,3\}$$
$$color^I = \{r,g,b\} \qquad assign^I = \{(1,r),(2,g),(3,b)\}$$

and $I \models \Delta$. Then, among other axioms, I satisfies

$$\forall X T \Big(T \in set_{asg}(X) \leftrightarrow \exists Y \left(T = tuple_2(X,Y) \wedge assign(X,Y) \wedge color(Y) \right) \Big)$$

and, thus, it follows that

$$set_{|asg/X|}(1)^I = \{(1,r)\} \qquad set_{|asg/X|}(2)^I = \{(2,g)\} \qquad set_{|asg/X|}(3)^I = \{(3,b)\}$$

Similarly, I satisfies

$$\forall X\, S\, \big(\text{Subset}(S, set_{|asg/X|}(X)) \rightarrow \mathit{FiniteCount}(S)\big)$$

and, thus, it follows that

$$count(set_{|asg/X|}(1))^I = 1 \qquad count(set_{|asg/X|}(2))^I = 1 \qquad count(set_{|asg/X|}(3))^I = 1$$

This standard interpretation also satisfies all formulas in $\text{COMP}_{\mathbf{p}}[\kappa\Pi]$. *Then,* $Ans(I)$ *(the set of atoms satisfied by I whose predicate symbol occurs in* $\mathbf{p}$*) is*

$$\{\, edge(1,2), edge(2,3), edge(3,1), vertex(1), vertex(2), vertex(3),\\ color(r), color(g), color(b), assign(1,r), assign(2,g), assign(3,b)\,\}$$

which corresponds to an answer set of Π.

8. Discussion and Conclusions

In this paper, we provided a characterization of the semantics of programs with aggregates that bypasses grounding. This is achieved by introducing a translation from logic programs to many-sorted first-order sentences together with an axiomatization in second-order logic. Interestingly, the introduced semantics coincide with the `ASP-Core-2` semantics (Calimeri et al., 2020) for programs that obey the restrictions (lack of recursive over aggregates and double negated literals) imposed by the `ASP-Core-2` standard. If we lift some restrictions and allow double negated literals and aggregates with recursion over negation, our semantics coincide with the semantics of the widely used solver CLINGO (Gebser et al., 2015). Furthermore, aggregates over infinite sets do not occur in practice (Calimeri et al., 2020) and, in this case, the second-order axiomatization can be replaced by first-order sentences.

Our work contributes to the understanding of aggregates in ASP and, in particular, it is the first to formalize that aggregates, as defined by the `ASP-Core-2` standard, agree with the practitioner's intuition that they represent functions applied to sets. It also provides an easy way to introduce new aggregate operations in the formalization, such as *product* or *average*, by simply adding the corresponding axioms for the new operation.

Our work also paves the way for the use of first-order theorem provers for reasoning about this class of programs, something that, to the best of our knowledge, was not possible before our characterization. A line of future work is directed to exploit this fact for the formal verification of ASP programs. The latest progress in this direction is represented by the ANTHEM system (Fandinno et al., 2020), and its extension, ANTHEM-P2P (Fandinno et al., 2023). These systems transform programs into their formula representations using a variant of the translation presented in Section 4, then capture the semantics of the corresponding programs using completion, as described in Section 7. The resolution theorem prover VAMPIRE (Kovács & Voronkov, 2013) uses the resulting completions to confirm the adherence of a program to a first-order specification (in the ANTHEM system), or to confirm the equivalent *external behavior* of two programs under a set of assumptions (in the ANTHEM-P2P system). Extending ANTHEM to programs with aggregates will require

combining our characterization of aggregates with the characterization of arithmetic expressions already present in ANTHEM, and provide a suitable axiomatization that characterizes standard interpretations. VAMPIRE already possesses the capacity to reason about integer arithmetic. Therefore, the main challenge is to provide VAMPIRE with an appropriate background theory of sets. Another open question is how to use first-order theorem provers to reason about non-tight programs, which is a matter of ongoing research.

Another future line of work is to extend our characterization to programs with positive recursion through aggregates. This will require different trade-offs to accommodate the different competing semantics. However, a characterization of aggregates without any restrictions on recursion would allow us to extend the definition of strong equivalence (Lifschitz, Pearce, & Valverde, 2001) to programs with aggregates. This is a powerful tool for simplifying and refactoring components of ASP programs in isolation, that is, without reference to the enclosing program. Importantly, strong equivalence does not require completion and, thus, it is not limited to tight programs.

Acknowledgments

We would like to thank Vladimir Lifschitz and the anonymous reviewers for their valuable feedback on multiple iterations of this project.

Appendix A. Proof of the Results

In this appendix, we provide formal support for the paper's claims. The appendix is organized as follows. Section A.1 is a proof of Theorem 1. This result establishes the relationship of our proposed definition of stable models (defined in terms of agg-interpretations) to gringo answer sets (defined in terms of a translation to infinitary propositional logic). This is a helpful intermediate step, but it is limited by how generous assumptions 5-10 of agg-interpretations are. Section A.2 justifies replacing these assumptions with axiomatizations and much more restrictive assumptions (standard interpretation assumptions 11-19). Sections A.3-A.6 prove that our proposed axioms correctly characterize the behavior of aggregate function symbols $\widehat{\texttt{count}}$, $\widehat{\texttt{sum}}$, $\widehat{\texttt{sum+}}$, $\widehat{\texttt{min}}$, and $\widehat{\texttt{max}}$. These results culminate in a proof of Theorem 3 (Section A.7), which connects our definition of answer sets (in terms of agg-interpretations) to their definition in terms of more restrictive standard interpretations. Finally, Section A.7 establishes that the second-order characterization of aggregate behavior can be replaced with a first-order axiomatization in the case of finite aggregates.

A.1 Proof of Theorem 1

We now provide a proof of the result relating the proposed semantics of programs with aggregates with the `Abstract Gringo` semantics. To do so, we first review earlier results on infinitary grounding and splitting.

A.1.1 INFINITARY GROUNDING

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.

Let $\mathbf{p},\mathbf{q}$ be a partition of the predicate symbols in the signature. Then, the *grounding of a first-order sentence with respect to an interpretation I and a set $\mathbf{p}$ of predicate symbols* is defined recursively as follows:

- $gr_I^{\mathbf{p}}(\bot) = \bot$;
- if formula is an atomic formula then
  - for $p \in \mathbf{p}$, $gr_I^{\mathbf{p}}(p(t_1,\ldots,t_k)) = p((t_1^I)^*,\ldots,(t_k^I)^*)$;
  - for $p \in \mathbf{q}$, $gr_I^{\mathbf{p}}(p(t_1,\ldots,t_k)) = \top$ if $p((t_1^I)^*,\ldots,(t_k^I)^*) \in I^{\mathbf{q}}$; and $gr_I^{\mathbf{p}}(p(t_1,\ldots,t_k)) = \bot$ otherwise;
  - $gr_I^{\mathbf{p}}(t_1 = t_2) = \top$ if $t_1^I = t_2^I$ and $\bot$ otherwise;
- $gr_I^{\mathbf{p}}(F \otimes G) = gr_I^{\mathbf{p}}(F) \otimes gr_I^{\mathbf{p}}(G)$ if $\otimes$ is $\wedge$, $\vee$, or $\rightarrow$;
- $gr_I^{\mathbf{p}}(\exists X\, F(X)) = \{gr_I^{\mathbf{p}}(F(u^*)) \mid u \in |I|^s\}^{\vee}$ if X is a variable of sort s;
- $gr_I^{\mathbf{p}}(\forall X\, F(X)) = \{gr_I^{\mathbf{p}}(F(u^*)) \mid u \in |I|^s\}^{\wedge}$ if X is a variable of sort s.

For a first-order theory Γ, we define $gr_I^{\mathbf{p}}(\Gamma) = \{gr_I^{\mathbf{p}}(F) \mid F \in \Gamma\}^{\wedge}$. The proof of the following Lemma is analogous to the proof of Theorem 5 by Truszczyński (2012). See Proposition 1 by Fandinno et al. (2020) for more details.

Lemma 1. *Let Γ be a first-order formula and I be some standard interpretation. Then I is a model of $\mathrm{SM}_{\mathbf{p}}[\Gamma]$ iff $Ans(I)$ is an FT-stable model of $gr_I^{\mathbf{p}}(\Gamma)$.*

A.1.2 SPLITTING THEOREM FOR INFINITARY PROPOSITIONAL FORMULAS

We start by recalling that the set of *strictly positive atoms* of an infinitary formula F, denoted $\mathrm{Pos}(F)$, is defined recursively:

- $\mathrm{Pos}(A) = \{A\}$ for an atom A,
- $\mathrm{Pos}(\mathscr{H}^{\wedge}) = \mathrm{Pos}(\mathscr{H}^{\vee}) = \bigcup_{H\in\mathscr{H}} \mathrm{Pos}(H)$,
- $\mathrm{Pos}(G \rightarrow H) = \mathrm{Pos}(H)$.

The set of *positive nonnegated atoms* and the set of *negative nonnegated atoms* of an infinitary formula F, denoted $\mathrm{Pnn}(F)$ and $\mathrm{Nnn}(F)$, are recursively defined:

- $\mathrm{Pnn}(A) = \{A\}$ for an atom A,
- $\mathrm{Pnn}(\mathscr{H}^{\wedge}) = \mathrm{Pnn}(\mathscr{H}^{\vee}) = \bigcup_{H\in\mathscr{H}} \mathrm{Pnn}(H)$,
- $\mathrm{Pnn}(G \rightarrow H) = \emptyset$ if H is $\bot$; and $\mathrm{Nnn}(G) \cup \mathrm{Pnn}(H)$ otherwise.
- $\mathrm{Nnn}(A) = \emptyset$ for an atom A,
- $\mathrm{Nnn}(\mathscr{H}^{\wedge}) = \mathrm{Nnn}(\mathscr{H}^{\vee}) = \bigcup_{H\in\mathscr{H}} \mathrm{Nnn}(H)$,
- $\mathrm{Nnn}(G \rightarrow H) = \emptyset$ if H is $\bot$; and $\mathrm{Pnn}(G) \cup \mathrm{Nnn}(H)$ otherwise.

The set of *rules* of an infinitary formula F, denoted $\text{Rules}(F)$, is defined as follows:

- $\text{Rules}(A) = \emptyset$ for an atom A,
- $\text{Rules}(\mathscr{H}^{\wedge}) = \text{Rules}(\mathscr{H}^{\vee}) = \bigcup_{H \in \mathscr{H}} \text{Rules}(H)$,
- $\text{Rules}(G \to H) = \{G \to H\} \cup \text{Rules}(H)$.

For any infinitary formula F and a set S of ground atoms, its *S-dependency graph* is a directed graph such that:

(a) its vertices are the ground atoms in S, and

(b) for every rule $(\mathit{Body} \to \mathit{Head})$ in $\text{Rules}(F)$, every atom $B \in \text{Pnn}(\mathit{Body})$ and every atom $H \in \text{Pos}(\mathit{Head})$, it includes the edge (H, B).

For an infinitary formula F and set S of ground atoms, by $\text{Choice}(F, S)$ we denote the conjunction of all disjunctions of form $A \vee \neg A$ for ground atoms occurring in F that do not belong to S. We say that a set $\mathscr{A}$ of ground atoms is an *S-infinitary stable model* of a formula F if it is an FT-stable model of $F \wedge \text{Choice}(F, S)$. Given a set S of ground atoms, a partition $\langle S_1, S_2 \rangle$ of S is *infinitely separable* with respect to an infinitary formula F if every infinite walk of its S-dependency graph visits either S_1 or S_2 finitely many times. Then, we have the following result (Harrison & Lifschitz, 2016, Infinitary Splitting Theorem).

Lemma 2. *Let F_1, F_2 be infinitary formulas and $\langle S_1, S_2 \rangle$ be a partition of some set S of atoms that is infinitely separable with respect to $F_1 \wedge F_2$. Let $\mathscr{A}$ be a set of atoms. If $S_2 \cap \text{Pos}(F_1) = \emptyset$ and $S_1 \cap \text{Pos}(F_2) = \emptyset$, then $\mathscr{A}$ is an S-infinitary stable model of $F_1 \wedge F_2$ iff it is both an S_1-infinitary stable model of F_1 and an S_2-infinitary stable model of F_2.*

A.1.3 AUXILIARY RESULTS

We now show how Lemma 2 can be used to turn any program without recursive aggregates into a program in which all aggregate atoms occur in the scope of negation.

Given a program Π and an aggregate atom A occurring in Π, we partition the predicate symbols occurring in Π into two sets $t(\Pi, A)$ and $b(\Pi, A)$ as follows. Let $b(\Pi, A)$ be the set of all predicate symbols p/n occurring in Π such that there is a path in the program's dependency graph from any predicate symbol occurring in A to p/n. Let $t(\Pi, A)$ be the set of all predicate symbols occurring in Π but those in $b(\Pi, A)$. Let $ba(\Pi, A)$ be the set of all ground atoms of form $p(\mathbf{t})$ such that p/n belongs to $b(\Pi, A)$ and $\mathbf{t}$ is an n-tuple of ground program terms. Let $br(\Pi, A)$ be the set of all rules of Π whose head contains a predicate symbol that belongs to $b(\Pi, A)$. Similarly, for $ta(\Pi, A)$ and $tr(\Pi, A)$.

Lemma 3. *For any program Π without positively recursive aggregates, pair $\langle ta(\Pi, A), ba(\Pi, A) \rangle$ is infinitely separable with respect to $\tau\Pi$.*

Proof. Let $S = ta(\Pi, A) \cup ba(\Pi, A)$ be the set of all ground atoms corresponding to the predicate symbols occurring in $\tau\Pi$. Suppose, for the sake of contradiction, that there is an infinite walk $A_1, A_2, \ldots$ that visits both $ta(\Pi, A)$ and $ba(\Pi, A)$ infinitely many times, that is, both $\{i \mid A_i \in ta(\Pi, A)\}$ and $\{i \mid A_i \in ba(\Pi, A)\}$ are infinite sets.

Take any $A_i \in ba(\Pi,A)$ and $A_j \in ta(\Pi,A)$ such that $j > i$. Note that such A_i and A_j must exist because the walk visits both sets infinitely many times. Let p_i and p_j be the predicate symbols occurring in A_i and A_j, respectively. Since there is a path from A_i to A_j in the S-dependency graph of $\tau\Pi$, there is a path from p_i to p_j in the program dependency graph of Π. Furthermore, by construction, A_i in $ba(\Pi,A)$ implies p_i in $b(\Pi,A)$, which in its turn implies that there is a path in the positive dependency graph from some predicate symbol q to p_i such that q occurs in A. These two facts together imply that there is a path in the positive dependency graph from some predicate symbol q to p_j. This implies that p_j belongs to $b(\Pi,A)$ and, thus, that A_j belongs to $ba(\Pi,A)$. This is a contradiction with the fact that $A_j \in ta(\Pi,A)$. □

Lemma 4. *Let Π be a program without positively recursive aggregates, A be an occurrence of some aggregate atom, and Π_b and Π_t be the sets of all rules whose head contains a predicate symbol in $b(\Pi,A)$ and $t(\Pi,A)$, respectively. Then, a set $\mathscr{A}$ of atoms is a gringo answer set of Π iff it is both a $ba(\Pi,A)$-infinitary stable model of $\tau\Pi_b$ and a $ta(\Pi,A)$-infinitary stable model of $\tau\Pi_t$.*

Proof. By definition, $\mathscr{A}$ is a gringo answer set of Π iff $\mathscr{A}$ is an FT-stable model of $\tau\Pi = \tau\Pi_b \wedge \tau\Pi_t$. From Lemma 3, it follows that $\langle ta(\Pi,A), ba(\Pi,A)\rangle$ is infinitely separable with respect to $\tau\Pi$. Furthermore, by construction we get $\mathrm{Pos}(\Pi_t) \subseteq ta(\Pi,A)$ and $\mathrm{Pos}(\Pi_b) \subseteq ba(\Pi,A)$ and that sets $ta(\Pi,A)$ and $ba(\Pi,A)$ are disjoint. Hence, $\mathrm{Pos}(\Pi_b) \cap ta(\Pi,A) = \emptyset$ and $ba(\Pi,A) \cap \mathrm{Pos}(\Pi_t) = \emptyset$ hold and the lemma statement follows now directly from Lemma 2. □

Lemma 5. *Let Π be a program and let Π' be the result of replacing some occurrence A of an aggregate atom by not not A. Let S be a set of ground atoms that contains no atom occurring in τA. Then, the S-infinitary stable models of $\tau\Pi$ and $\tau\Pi'$ coincide.*

Proof. Recall that, by definition, a set $\mathscr{A}$ of atoms is an S-infinitary stable model of $\tau\Pi$ iff $\mathscr{A}$ is an FT-stable model of $\tau\Pi \wedge \mathrm{Choice}(\tau\Pi, S)$. Furthermore, since no atom occurring in τA belongs to S, it follows that the excluded middle axiom $B \vee \neg B$ belongs to $\mathrm{Choice}(\tau\Pi, S)$ for every atom B occurring in τA. Hence, we can replace τA by $\neg\neg\tau A$ without changing the FT-stable models. □

Lemma 6. *Let Π be a program without positively recursive aggregates and let Π' be the result of replacing each aggregate atom A not in the scope of negation by not not A. Then, the gringo answer sets of Π and Π' coincide.*

Proof. We just need to prove it for a single occurrence A of some aggregate atom and the result follows then by induction in the number of occurrences of aggregate atoms not in the scope of negation. Let R be the rule of Π containing occurrence A, rule R' be the result of replacing occurrence A not in the scope of negation by *not not* A in R and $\Pi' = (\Pi \setminus \{R\}) \cup \{R'\}$ be the result of replacing occurrence A by *not not* A in Π. Let Π_b and Π_t be the set of all rules whose head contains a predicate symbol in $b(\Pi,A)$ and $t(\Pi,A)$, respectively. Π'_b and Π'_t are constructed similarly. Note that rule R belongs to Π_t because aggregates in Π are not positively recursive and that this implies that $\Pi_b = \Pi'_b$. Note also that $ba(\Pi,A) = ba(\Pi',A)$ and $ta(\Pi,A) = ta(\Pi',A)$. Then, from Lemmas 4 and 5, it follows that a set of atoms $\mathscr{A}$ is a gringo answer set of Π
iff $\mathscr{A}$ is both a $ba(\Pi,A)$-infinitary stable model of $\tau\Pi_b$ and a $ta(\Pi,A)$-infinitary stable model of $\tau\Pi_t$
iff $\mathscr{A}$ is both a $ba(\Pi',A)$-infinitary stable model of $\tau\Pi'_b$ and a $ta(\Pi',A)$-infinitary stable model of $\tau\Pi'_t$
iff $\mathscr{A}$ is a gringo answer set of Π'. □

Theorem 1 follows directly from Lemma 1 and the following auxiliary results.

Lemma 7. *Let I be an agg-interpretation and op be either count, sum, sum$^+$, min, or max. Then, I satisfies $op(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u$ iff $Ans(I)$ satisfies*

$$\bigwedge_{\Delta\in\chi}\left(\bigwedge_{\mathbf{y}\in\Delta}\mathbf{l}^{\mathbf{XY}}_{\mathbf{xy}} \rightarrow \bigvee_{\mathbf{y}\in\Psi_{E^{\mathbf{X}}_{\mathbf{x}}}\setminus\Delta}\mathbf{l}^{\mathbf{XY}}_{\mathbf{xy}}\right) \tag{43}$$

where χ is the set of subsets Δ of $\Psi_{E^{\mathbf{X}}_{\mathbf{x}}}$ that do not justify aggregate atom $op\{E^{\mathbf{X}}_{\mathbf{x}}\} \prec u$ and $\mathbf{Y}$ is the list of variables occurring in E that do not occur in $\mathbf{X}$.

Proof. Let $\Delta_I = \{\, \mathbf{y} \in \Psi_{E^{\mathbf{X}}_{\mathbf{x}}} \mid I \models \mathbf{l}^{\mathbf{XY}}_{\mathbf{xy}} \,\}$ and H_I be the formula

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

Then, $Ans(I) \not\models H_I$ 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

$$\begin{aligned}
Ans(I) \models (43) &\text{ iff } H_I \text{ is not a conjunctive term of (43)}\\
&\text{iff } \Delta_I \text{ justifies } op(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u\\
&\text{iff } I \models op([\Delta_I]^*) \prec u\\
&\text{iff } I \models op(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u
\end{aligned}$$

Lemma 8. *Let Π be a program in which all aggregate atoms occur in the scope of negation and let I be an agg-interpretation. Then, $Ans(I)$ is an FT-stable model of $\tau\Pi$ iff it is an FT-stable model of $gr^{\mathbf{p}}_I(\kappa\Pi)$.*

Proof. Recall that comparisons do not belong to $\mathbf{p}$ in the definition of the stable models of a program. Then, it is easy to see that $\tau\Pi$ can be obtained from $gr^{\mathbf{p}}_I(\kappa\Pi)$ by replacing each occurrence of $\neg op(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u$, where $op \in \{count, sum, sum^+, min, max\}$, by $\neg$(43): 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

$$\left(\neg op(set_{|E/\mathbf{X}|}(\mathbf{x})) \prec u\right)^{Ans(I)} \quad = \quad (\neg(43))^{Ans(I)}$$

For this it is enough to show that I satisfies $op(set_{|E/\mathbf{x}|}(\mathbf{X})) \prec \mathbf{u}$ iff $Ans(I)$ satisfies (43), which follows from Lemma 7. □

A.1.4 PROOF OF THE THEOREM

Proof of Theorem 1. Let Π be a program without positively recursive aggregates and let Π' be the result of replacing each aggregate atom A not in the scope of negation by *not not A*. Let I be an agg-interpretation and $\mathscr{A} = Ans(I)$. Then,

$\mathscr{A}$ is a gringo answer set of Π
iff $\mathscr{A}$ is a gringo answer set of Π' (Lemma 6)
iff $\mathscr{A}$ is an FT-stable model of $\tau\Pi'$ (by definition)
iff $\mathscr{A}$ is an FT-stable model of $gr^{\mathbf{p}}_I(\kappa\Pi')$ (Lemma 8)
iff $I \models \mathrm{SM}_{\mathbf{p}}[\kappa\Pi']$ and $\mathscr{A} = Ans(I)$ (Lemma 1)
iff $I \models \mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ and $\mathscr{A} = Ans(I)$ (see below)

iff $\mathscr{A}$ is an answer set of Π. (by definition)
For the missing step, recall that comparison symbols do not belong to $\mathbf{p}$. Therefore, we can replace any atom of form $t \prec t'$ with $\prec$ a comparison symbol by $\neg\neg(t \prec t')$ without changing the models of $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$, that is, $I \models \mathrm{SM}_{\mathbf{p}}[\kappa\Pi']$ iff $I \models \mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$. □

A.2 Proof of Propositions 1-2

Lemma 9. *Let E be an aggregate element of the form* (2) *with free variables* $\mathbf{V}$ *and bound variables* $\mathbf{W}$ *and let I be a standard interpretation. Let* $\mathbf{v}$ *be a list of ground terms of sort s_{prg} of the same length as* $\mathbf{V}$*, $l'_i = (l_i)^{\mathbf{V}}_{\mathbf{v}}$ and $t'_i = (t_i)^{\mathbf{V}}_{\mathbf{v}}$. Then, I satisfies*

$$\forall T \big(T \in set_{|E|}(\mathbf{v}) \leftrightarrow \exists \mathbf{W} \, (T = tuple_m(t'_1, \ldots, t'_m) \wedge l'_1 \wedge \cdots \wedge l'_n) \big) \tag{44}$$

iff $set_{|E|}(\mathbf{v})^I$ 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 $t''_i = (t'_i)^{\mathbf{W}}_{\mathbf{w}}$ and $l''_i = (l'_i)^{\mathbf{W}}_{\mathbf{w}}$ and $\mathbf{w}$ *a list of ground terms of sort s_{prg} of the same length as* $\mathbf{W}$.

Proof. Left-to-right. Assume that I satisfies (44). Pick any domain element d_{tuple} of sort s_{tuple}. Since I is a standard interpretation, d_{tuple} belongs to $set_{|E|}(\mathbf{v})^I$ iff I satisfies $\in (d^*_{tuple}, set_{|E|}(\mathbf{v}))$. Furthermore, since I satisfies (44), the latter holds iff there is a list $\mathbf{c}$ of domain elements of sort s_{prg} such that

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

and I satisfies $l''_1 \wedge \cdots \wedge l''_n$ with $\mathbf{w} = \mathbf{c}^*$.

Right-to-left. Assume that $set_{|E|}(\mathbf{v})^I$ 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 $\mathbf{w}$ of ground terms of sort s_{prg} of the same length as $\mathbf{W}$. We need to show that I satisfies (44). Pick any domain element d_{tuple} of sort s_{tuple} and we will show that I satisfies

$$d^*_{tuple} \in sets_{|E|}(\mathbf{d}^*) \leftrightarrow \exists \mathbf{W} \left(d^*_{tuple} = tuple_m(t'_1, \ldots, t'_m) \wedge l'_1 \wedge \cdots \wedge l'_n \right) \tag{45}$$

Since I is a standard interpretation, it follows that
I satisfies $\in (d^*_{tuple}, set_{|E|}(\mathbf{d}^*))$
iff d_{tuple} belongs to $set_{|E|}(\mathbf{d}^*)^I$
iff there exists $\mathbf{w}$ such that $d_{tuple} = \langle (t''_1)^I, \ldots, (t''_m)^I \rangle$ and I satisfies $l''_1 \wedge \cdots \wedge l''_n$
iff I satisfies $\exists \mathbf{W} \, (d^*_{tuple} = tuple_m(t'_1, \ldots, t'_m) \wedge l'_1 \wedge \cdots \wedge l'_n)$. □

Proof of Proposition 1. Pick any list $\mathbf{c}$ of domain elements of sort s_{prg} and let $\mathbf{v} = \mathbf{c}^*$. Then, the result follows directly from Lemma 9. □

Lemma 10. *Let I be a standard interpretation, d_{set}, d_{tuple}, and e_{set} be domain elements of sorts s_{set}, s_{tuple}, and s_{set}, respectively. Then,*

$$d_{set} \setminus \{d_{tuple}\} = e_{set} \tag{46}$$

holds iff formula

$$\forall T' \big(T' \in e^*_{set} \leftrightarrow (T' \in d^*_{set} \wedge T \neq d^*_{tuple}) \big) \tag{47}$$

is satisfied by I.

Proof. Left-to-right. Assume that (46) holds and pick an arbitrary domain element c_{tuple} of sort s_{tuple}. We need to show that

$$c^*_{tuple} \in e^*_{set} \leftrightarrow (c^*_{tuple} \in d^*_{set} \wedge c^*_{tuple} \neq d^*_{tuple}) \tag{48}$$

is satisfied by I. Since I is a standard interpretation, it follows that

I satisfies $c^*_{tuple} \in e^*_{set}$
iff $c_{tuple} \in e_{set}$
iff $c_{tuple} \in d_{set} \setminus \{d_{tuple}\}$
iff $c_{tuple} \in d_{set}$ and $c_{tuple} \notin \{d_{tuple}\}$
iff $c_{tuple} \in d_{set}$ and $c_{tuple} \neq d_{tuple}$
iff I satisfies $c^*_{tuple} \in d^*_{set} \wedge c^*_{tuple} \neq d^*_{tuple}$.

Right-to-left. Assume that (47) holds. We will show that (46) holds as well. Pick any element c_{tuple} from the tuple domain. Then, since I is a standard interpretation, it follows that

$c_{tuple} \in e_{set}$
iff I satisfies $c^*_{tuple} \in e^*_{set}$
iff I satisfies $c^*_{tuple} \in d^*_{set} \wedge c^*_{tuple} \neq d^*_{tuple}$
iff $c_{tuple} \in d_{set}$ and $c_{tuple} \neq d_{tuple}$
iff $c_{tuple} \in d_{set}$ and $c_{tuple} \notin \{d_{tuple}\}$
iff $c_{tuple} \in d_{set} \setminus \{d_{tuple}\}$. Therefore, (46) holds. □

Lemma 11. *Let I be a standard interpretation. Then, I satisfies condition 17 of agg-interpretations iff it satisfies sentence* (17)*.*

Proof. Left to right. Assume that I satisfies condition 17 of aggregate interpretations. We now show that I is a model of (17). Pick arbitrary domain elements $d_{set}, e_{set} \in |I|^{s_{set}}$ and $d_{tuple} \in |I|^{s_{tuple}}$. Then,

I satisfies $rem(d^*_{set}, d^*_{tuple}) = e^*_{set}$
iff $(rem(d^*_{set}, d^*_{tuple}))^I = e_{set}$
iff $d_{set} \setminus \{d_{tuple}\} = e_{set}$ (condition 17)
iff I satisfies (47) (Lemma 10).
Therefore, I satisfies (17).

Right to left. Assume now that I is a model of (17). We now show that I satisfies condition 17 of aggregate interpretations. Pick any term t_{set} of sort s_{set} and any term t_{tuple} of sort s_{tuple}. Let $t^I_{set} = d_{set}$ and $t^I_{tuple} = d_{tuple}$. Then, by definition,

$$\begin{aligned} rem(t_{set}, t_{tuple})^I &= rem(t^I_{set}, t^I_{tuple})^I \\ &= rem(d_{set}, d_{tuple})^I = e_{set} \end{aligned} \tag{49}$$

for some set $e_{set} \in |I|^{s_{set}}$ (given that I is a standard interpretation, e is a set of elements of $|I|^{s_{tuple}}$). We need to show that (46) holds. Note that, since I is a model of (17), it satisfies (47) and the result follows immediately by Lemma 10. □

Lemma 12. *Let I be a standard interpretation. Then, I satisfies condition 18 of agg-interpretations iff it satisfies all sentences of the form* (18)*.*

Proof. Left to Right. Assume I satisfies condition 18. Choose arbitrary domain elements $d_1, \ldots, d_k$ of sort s_{prg} and take term $t_{tuple} = tuple_k(d^*_1, \ldots, d^*_k)$. By condition 14 of standard interpretations, it

follows that $t^I_{tuple} = d_{tuple}$. Then, by assumption, we get that

$$first(t_{tuple})^I = first(tuple_k(d_1^*,\ldots,d_k^*))^I = first^I(\langle d_1,\ldots,d_k\rangle) = d_1.$$

Therefore,

$$I \models \forall X_1 \ldots X_k \big(first(tuple_k(X_1,\ldots,X_k)) = X_1\big)$$

Right to Left. Assume $I \models \forall X_1 \ldots X_k \big(first(tuple_k(X_1,\ldots,X_k)) = X_1\big)$ for any k. Choose arbitrary domain elements $d_1,\ldots,d_k$ of sort s_{prg}. Then

$$I \models first(tuple_k(d_1^*,\ldots,d_k^*)) = d_1^*$$

By condition 14 of standard interpretations, $tuple_k(d_1^*,\ldots,d_k^*)^I = \langle d_1,\ldots,d_k\rangle$, for some domain element d_{tuple} of sort s_{tuple}. Thus, I satisfies condition 18. □

Lemma 13. *Let I be a standard interpretation and $d_1, d_2 \ldots, d_n$ be domain elements of sort s_{prg}. Then, I satisfies*

$$(\exists N\, d_1^* = N) \rightarrow weight(tuple_k(d_1^*, d_2^* \ldots, d_n^*)) = d_1^* \tag{50}$$
$$(\neg\exists N\, d_1^* = N) \rightarrow weight(tuple_k(d_1^*, d_2^* \ldots, d_n^*)) = 0. \tag{51}$$

iff $weight(tuple_k(d_1, d_2 \ldots, d_n))^I$ is the weight of $tuple_k(d_1, d_2 \ldots, d_n)$ in the sense of condition 19.

Proof. Assume first that d_1 is an integer. Then, I does not satisfy formula $\neg\exists N\, d_1^* = N$ and, thus, it satisfies (51). Hence, it is enough to show that I satisfies

$$weight(tuple_k(d_1^*, d_2^* \ldots, d_n^*)) = d_1^*. \tag{52}$$

iff $weight(tuple_k(d_1, d_2 \ldots, d_n))^I$ is the weight of $tuple_k(d_1^*, d_2^* \ldots, d_n^*)^I$. Since I is a standard interpretation, it follows that $tuple_k(d_1^*, d_2^* \ldots, d_n^*)^I$ is $\langle d_1, d_2 \ldots, d_n\rangle$. Since d_1 is an integer, the weight of $tuple_k(d_1^*, d_2^* \ldots, d_n^*)^I$ is d_1. Hence, the above holds.

Assume now that d_1 is not an integer. Then, I does not satisfy $\exists N\, d_1^* = N$ and, thus, it satisfies (50). Hence, it is enough to show that I satisfies

$$weight(tuple_k(d_1^*, d_2^* \ldots, d_n^*)) = \overline{0}. \tag{53}$$

iff $weight(tuple_k(d_1, d_2 \ldots, d_n))^I$ is the weight of $tuple_k(d_1^*, d_2^* \ldots, d_n^*)^I$. Indeed, since I is a standard interpretation, $tuple_k(d_1^*, d_2^* \ldots, d_n^*)^I$ is $\langle d_1, d_2 \ldots, d_n\rangle$. Since d_1 is not integer, the weight of $tuple_k(d_1^*, d_2^* \ldots, d_n^*)^I$ is 0. □

Lemma 14. *Let I be a standard interpretation. Then, I satisfies condition 19 iff it satisfies all sentences of form (19-20).*

Proof. Left-to-right. From Lemma 13, I satisfies condition 19 of aggregate interpretations iff I satisfies sentences

$$\forall X_1 \forall X_2 \ldots \forall X_n \big((\exists N\, X_1 = N) \rightarrow weight(tuple_k(X_1, X_2, \ldots, X_n)) = X_1\big) \tag{54}$$
$$\forall X_1 \forall X_2 \ldots \forall X_n \big((\neg\exists N\, X_1 = N) \rightarrow weight(tuple_k(X_1, X_2, \ldots, X_n)) = 0\big).$$

Now, it is enough to note that (19) and (54) are equivalent in first order logic. □

Proof of Proposition 2. This proposition is just a summary of Lemmas 11, 12 and 14. □

A.3 Proof of Proposition 3

Lemma 15. *Let I be a standard interpretation and t_{set} be a term of sort s_{set}. Then, I satisfies formula $Finite(t_{set})$ iff set t^I_{set} is finite, that is, iff there is a bijection between this set and a set of natural numbers of form $\{i \in \mathbb{N} \mid i \leq n\}$ for some natural number n.*

Proof. First note that $I \models Finite(t_{set})$
iff
$$(\exists f\,(Injective(f, t_{set}) \wedge \exists NImage(f, t_{set}, 0, N)))^I = \mathbf{true}$$
iff there exists a function f such that
$$Injective(f, t_{set})^I = \mathbf{true}$$
and
$$(\exists NImage(f, t_{set}, 0, N))^I = \mathbf{true}$$
Furthermore, $Injective(f, t_{set})^I = \mathbf{true}$
iff arbitrary domain elements $v_1 \in |I|^{s_{tuple}}, v_2 \in |I|^{s_{tuple}}$ satisfy
$$(v_1^* \in t_{set}^* \wedge v_2^* \in t_{set}^* \wedge f(v_1^*) = f(v_2^*) \rightarrow v_1^* = v_2^*)^I = \mathbf{true}$$
iff for arbitrary domain elements $v_1 \in |I|^{s_{tuple}}, v_2 \in |I|^{s_{tuple}}$,
  if $v_1 \in t^I_{set}$ and $v_2 \in t^I_{set}$ and $f^I(v_1) = f^I(v_2)$
  then $v_1 = v_2$
iff the restriction of f^I to t^I_{set} is an injective function.

Similarly, $(\exists NImage(f, t_{set}, 0, N))^I = \mathbf{true}$
iff there exists a natural number m such that $Image(f, t_{set}, 0, m)^I = \mathbf{true}$
iff there exists a natural number m such that, for an arbitrary domain element $d_{tuple} \in |I|^{s_{tuple}}$, m satisfies
$$(d^*_{tuple} \in t_{set} \rightarrow 0 \leq f(d^*_{tuple}) \wedge f(d^*_{tuple}) \leq m)^I = \mathbf{true}$$
iff there exists a natural number m such that, for arbitrary domain elements $d_{tuple} \in |I|^{s_{tuple}}$,
  if $d_{tuple} \in t^I_{set}$ then $0 \leq f^I(d_{tuple}) \leq m$
iff there exists a natural number m such that every $d_{tuple} \in t^I_{set}$ satisfies $0 \leq f^I(d_{tuple}) \leq m$.

That is, $Finite(t_{set})^I = \mathbf{true}$ iff there is a natural number m and a function $f : t^I_{set} \longrightarrow \{i \in \mathbb{N} \mid i \leq m\}$ that is injective. Hence, we need to prove that the following two statements are equivalent:

1. there is a natural number m and a
   function $f : t^I_{set} \longrightarrow \{i \in \mathbb{N} \mid i \leq m\}$ that is injective, and

2. there is a natural number n and a
   function $g : t^I_{set} \longrightarrow \{i \in \mathbb{N} \mid i \leq n\}$ that is bijective.

It is clear the the latter implies the former, so we only need to prove that the former implies the latter. We proceed by induction in the number of elements l in $\{i \in \mathbb{N} \mid i \leq m\}$ that are not in the image of f.
Base case. If $l = 0$, then f is a bijection and we just define g and n as f and m, respectively.
Induction step. Otherwise, there exists some natural number $i \leq m$ such that $f(d_{tuple}) \neq i$ for all d_{tuple} in t^I_{set}. We define natural number n' as $m - 1$ and function g' as follows:

- $g'(d_{tuple}) \stackrel{\text{def}}{=} f(d_{tuple})$ for each $d_{tuple} \in t^I_{set}$ such that $f(d_{tuple}) \leq n'$;
- for each $d_{tuple} \in t^I_{set}$ such that $f(d_{tuple}) = m > n'$, we define $g'(d_{tuple}) \stackrel{\text{def}}{=} i$.

Since f is injective, function g' is also injective. Furthermore, $g'(d_{tuple}) \leq n'$ holds for every d_{tuple} in t^I_{set}. Note that, if $i > n'$, then every $d_{tuple} \in t^I_{set}$ satisfies $f(d_{tuple}) \leq n'$. Since there are strictly less elements in $\{i \in \mathbb{N} \mid i \leq n'\}$ that are not in the image of g' than l, the statement follows by the induction hypothesis. □

Lemma 16. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation. Let t_{set} be a term of sort s_{set} such that t^I_{set} contains finitely many and at least one tuple and such that I satisfies*

$$count(d^*_{set})^I \text{ is the cardinality of } d_{set} \tag{55}$$

for every domain element d_{set} of sort s_{set} which has strictly less tuples than t^I_{set}. Then, I satisfies formula

$$\forall T\big(T \in t_{set} \rightarrow \exists N\big(count(rem(t_{set}, T)) = N \wedge count(t_{set}) = N+1\big)\big) \tag{56}$$

iff

$$count(t_{set})^I \text{ is the cardinality of } t^I_{set} \tag{57}$$

Proof. Let d_{set} be the set obtained by removing some arbitrary domain element $d_{tuple} \in t^I_{set}$ from t^I_{set}. Note that, by assumption, t^I_{set} has at least one tuple. Note also that, by assumption t^I_{set} has finitely many tuples and, thus, d_{set} is finite. Therefore, its cardinality is a natural number n, that is, $count(d^*_{set})^I = n$ and the cardinality of t^I_{set} is $n+1$. Furthermore, since I is a standard interpretation and d_{tuple} belongs to t^I_{set}, it follows that I satisfies $d^*_{tuple} \in t_{set}$. Hence, it is enough to show that I also satisfies formula

$$\exists N\big(count(rem(t_{set}, d^*_{tuple})) = N \wedge count(t_{set}) = N+1\big) \tag{58}$$

iff $count(t_{set})^I = n+1$. Note also that, since I satisfies condition 17, it follows that

$$rem(t_{set}, d^*_{tuple})^I = t^I_{set} \setminus \{d_{tuple}\} = d_{set}$$

and, thus, formula (58) can be rewritten as

$$\exists N\,(count(d^*_{set}) = N \wedge count(t_{set}) = N+1) \tag{59}$$

Hence, it is enough to show that I also satisfies formula (59) iff $count(t_{set})^I = n+1$. This immediately follows by observing that $count(d^*_{set})^I = n$. □

Lemma 17. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation and sentence* (21)*. Let t_{set} be a term of sort s_{set} such that t^I_{set} contains finitely many tuples. Then, I satisfies formula*

$$\forall T\big(T \in t_{set} \rightarrow \exists N\big(count(rem(t_{set}, T)) = N \wedge count(t_{set}) = N+1\big)\big) \tag{60}$$

for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set} iff

$$count(d^*_{set})^I \text{ is the cardinality of } d_{set} \tag{61}$$

for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set}.

Proof. The proof follows by induction assuming that the lemma statement holds for all domain elements that have strictly fewer tuples.

Base case. In the case that d_{set} contains no tuples, since I is a standard interpretation it follows that the antecedent of (60) is never satisfied, which means I satisfies (60). By condition 15 of standard interpretations and by the fact that I satisfies sentence (21), $count(t_{set})^I = 0$ and so the lemma statement holds. Note that t^I_{set} is the only subset of itself as we assume it to be empty.

The *induction step* follows directly from Lemma 16. □

Lemma 18. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation and sentence* (21)*. Then, I satisfies sentence* (22) *iff*

$$count(d^*_{set})^I \text{ is the cardinality of } d_{set} \tag{62}$$

for every domain element d_{set} of sort s_{set} that contains a finite number of tuples.

Proof. Left-to-right. Assume I satisfies sentence (22) and choose arbitrary domain element $d_{set} \in |I|^{s_{set}}$ with a finite number of tuples. Since d_{set} is finite, it follows that each e_{set} that is a subset of d_{set} is also finite. By Lemma 15, this implies that I satisfies $Finite(e^*_{set})$ for each e_{set} that is a subset of d_{set}. Furthermore, since I satisfies sentence (22), it follows that any e_{set} satisfying $Finite(e^*_{set})$ also satisfies $FiniteCount(e^*_{set})$. From Lemma 17, this, plus the fact that I satisfies sentence (21), implies that $count(e^*_{set})^I$ is the cardinality of e_{set} for each e_{set}. In particular, this implies that I satisfies (62).

Right-to-Left. Assume $count(d^*_{set})^I$ is the cardinality of d_{set} for an arbitrary domain element d_{set} of sort s_{set}. Now assume that I satisfies $Finite(d^*_{set})$. By Lemma 15, d_{set} contains a finite number of tuples and thus, every subset e_{set} of d_{set} also contains a finite number of tuples. By the lemma assumption, this implies that

$$count(e^*_{set})^I \text{ is the cardinality of } e_{set}$$

for every domain element e_{set} of sort s_{set} which is a subset of d_{set}. From Lemma 17, this implies that I satisfies formula

$$\forall T\big(T \in e^*_{set} \rightarrow \exists N\big(count(rem(e^*_{set}, T)) = N \wedge count(e^*_{set}) = N+1\big)\big) \tag{63}$$

for every subset e_{set} of d_{set}. In particular, this implies that I satisfies $FiniteCount(d^*_{set})$ and, thus, that I satisfies (22). □

Proof of Proposition 3. *Left-to-Right.* Assume that I satisfies condition 6 of agg-interpretations and we will show that I satisfies sentences (21-23). Since I is a standard interpretation that satisfies condition 6, it follows that I interprets $count(t_{set})$ as $\widehat{\texttt{count}}(t^I_{set})$. This also implies that $\overline{\emptyset}^I = \emptyset$ and, thus, $count(\emptyset)^I = 0$. Therefore, I satisfies (21).

Let us now show that I satisfies (22). Pick an arbitrary domain element d^*_{set} of sort s_{set} and assume that I satisfies $Finite(d^*_{set})$. We need to show that I satisfies $FiniteCount(d^*_{set})$. From Lemma 15, it follows that d_{set} is finite. Recall that, by definition, $\widehat{\texttt{count}}(d_{set})$ is the cardinality of d_{set}. From Lemma 18, this implies that I satisfies $FiniteCount(d^*_{set})$ and so sentence (22) holds.

Finally, let us now show that I satisfies (23). Pick an arbitrary domain element d_{set} of sort s_{set} and assume that I does not satisfy $Finite(d^*_{set})$. By Lemma 15, this implies that d_{set} is infinite. Since I satisfies condition 6, it follows that I satisfies $count(d^*_{set}) = sup$. Therefore, I satisfies (23).

Right-to-left. Assume that I satisfies sentences (21-23) and we will show that it satisfies condition 6 of agg-interpretations. Pick any term t_{set} of sort s_{set}. If t_{set}^I is infinite, then (23) implies that $count(t_{set})^I = sup$. If t_{set}^I is finite, then by Lemma 18 and the fact that I satisfies (22) it follows that $count(t_{set})^I$ is the cardinality of t_{set}^I. Thus, interpretation I satisfies condition 6. □

A.4 Proof of Proposition 4

Lemma 19. *Let I be a standard interpretation and t_{set} be a term of sort s_{set}. Then, I satisfies formula $FiniteWeight(t_{set})$ iff set $\{d \in t_{set}^I \mid I \models weight(d^*) \neq 0\}$ is finite, that is, iff there is a bijection between this set and a set of natural numbers of form $\{i \in \mathbb{N} \mid i \leq n\}$ for some natural number n.*

Proof. First note that $I \models FiniteWeight(t_{set})$
iff

$$(\exists f\big(InjectiveWeight(f,t_{set}) \wedge \exists NImageWeight(f,t_{set},0,N)\big))^I = \textbf{true}$$

iff there exists a function f' such that

$$InjectiveWeight(f',t_{set})^I = \textbf{true}$$

and

$$(\exists NImageWeight(f',t_{set},0,N))^I = \textbf{true}$$

Furthermore, $InjectiveWeight(f',t_{set})^I = \textbf{true}$
iff arbitrary domain elements $v_1 \in |I|^{s_{tuple}}$ and $v_2 \in |I|^{s_{tuple}}$ satisfy

$$\begin{array}{l}(v_1^* \in t_{set}^* \wedge v_2^* \in t_{set}^* \wedge \\ \quad weight(v_1^*) \neq 0 \wedge weight(v_2^*) \neq 0 \wedge \\ \quad f'(v_1^*) = f'(v_2^*) \rightarrow v_1^* = v_2^*)^I = \textbf{true}\end{array} \tag{64}$$

iff for arbitrary domain elements $v_1 \in |I|^{s_{tuple}}, v_2 \in |I|^{s_{tuple}}$
if $v_1 \in t_{set}^I$ and $v_2 \in t_{set}^I$ and the weights of v_1 and v_2 are both non-zero and $f^I(v_1) = f^I(v_2)$, then $v_1 = v_2$
iff the restriction of f'^I to the set of elements of t_{set}^I with non-zero weights is an injective function.

Similarly, $(\exists NImageWeight(f',t_{set},0,N))^I = \textbf{true}$
iff there exists a natural number m such that

$$ImageWeight(f',t_{set},0,m)^I = \textbf{true}$$

iff there exists a natural number m such that, for arbitrary domain element $d_{tuple} \in |I|^{s_{tuple}}$, m satisfies

$$(d_{tuple}^* \in t_{set}^* \wedge weight(d_{tuple}^*) \neq 0 \rightarrow 0 \leq f(d_{tuple}^*) \wedge f(d_{tuple}^*) \leq m)^I = \textbf{true} \tag{65}$$

iff there exists a natural number m such that, for arbitrary domain element $d_{tuple} \in |I|^{s_{tuple}}$,

if $d_{tuple} \in t^I_{set}$ and the weight of d_{tuple} is non-zero,
then $0 \leq f'^I(d_{tuple}) \leq m$
iff there exists a natural number m s.t. every $d_{tuple} \in t^I_{set}$ with non-zero weight satisfies $0 \leq f'^I(d_{tuple}) \leq m$.

Hence, $I \models \mathit{FiniteWeight}(t_{set})$ iff there exists a natural number m and an injective function f from the set $\{d \in t^I_{set} \mid I \models \mathit{weight}(d^*) \neq 0\}$ (the set of elements in t^I_{set} with non-zero weights) to $\{i \in \mathbb{N} \mid i \leq m\}$. As was the case with Lemma 15, we need to prove that the following two statements are equivalent (omitting the trivial right to left direction):

1. there is a natural number m and function $f : \{d \in t^I_{set} \mid I \models \mathit{weight}(d^*) \neq 0\} \longrightarrow \{i \in \mathbb{N} \mid i \leq m\}$ that is injective, and
2. there is a natural number n and function $g : \{d \in t^I_{set} \mid I \models \mathit{weight}(d^*) \neq 0\} \longrightarrow \{i \in \mathbb{N} \mid i \leq n\}$ that is bijective.

We proceed by induction in the number of elements l in $\{i \in \mathbb{N} \mid i \leq m\}$ that are not in the image of f.
Base case. If $l = 0$, then f is a bijection and we just define g and n as f and m, respectively.
Induction step. Otherwise, there exists some natural number $i \leq m$ such that $f(d_{tuple}) \neq i$ for all d_{tuple} in $\{d \in t^I_{set} \mid I \models \mathit{weight}(d^*) \neq 0\}$. We define natural number n' as $m-1$ and function g' as follows:

- $g'(d_{tuple}) \stackrel{\text{def}}{=} f(d_{tuple})$ for each $d_{tuple} \in \{d \in t^I_{set} \mid I \models \mathit{weight}(d^*) \neq 0\}$ such that $f(d_{tuple}) \leq n'$;
- for each $d_{tuple} \in \{d \in t^I_{set} \mid I \models \mathit{weight}(d^*) \neq 0\}$ such that $f(d_{tuple}) > n'$, we define $g'(d_{tuple}) \stackrel{\text{def}}{=} i$.

Since f is injective, function g' is also injective. Furthermore, $g'(d_{tuple}) \leq n'$ holds for every d_{tuple} in $\{d \in t^I_{set} \mid I \models \mathit{weight}(d^*) \neq 0\}$. If $i > n'$, then every $d_{tuple} \in \{d \in t^I_{set} \mid I \models \mathit{weight}(d^*) \neq 0\}$ satisfies $f(d_{tuple}) \leq n'$. Since there are strictly less elements in $\{i \in \mathbb{N} \mid i \leq n'\}$ that are not in the image of g' than l, the statement follows by the induction hypothesis. □

Lemma 20. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation. Let t_{set} be a term of sort s_{set} such that t^I_{set} contains finitely many and at least one tuple with non-zero weight and such that I satisfies*

$$\mathit{sum}(d^*_{set})^I \;=\; \sum\{\mathit{weight}(d^*_{tuple})^I \mid d_{tuple} \in d_{set} \text{ and } \mathit{weight}(d^*_{tuple})^I \neq 0\} \tag{66}$$

for every domain element d_{set} of sort s_{set} which has strictly less non-zero tuples than t^I_{set}. Then, I satisfies formula

$$\forall T\big(T \in t_{set} \wedge \mathit{weight}(T) \neq 0 \rightarrow \exists N\big(\mathit{sum}(\mathit{rem}(t_{set}, T)) = N \wedge \mathit{sum}(t_{set}) = N + \mathit{weight}(T)\big)\big) \tag{67}$$

iff

$$\mathit{sum}(t_{set})^I \;=\; \sum\{\mathit{weight}(d^*_{tuple})^I \mid d_{tuple} \in t^I_{set} \text{ and } \mathit{weight}(d^*_{tuple})^I \neq 0\} \tag{68}$$

Proof. Let d_{set} be the set obtained by removing some arbitrary domain element d_{tuple} that belongs to t^I_{set} from t^I_{set} such that d_{tuple} has non-zero weight. Note that, by assumption, t^I_{set} has at least one non-zero weight tuple. Note also that, by assumption t^I_{set} has finitely many non-zero weight tuples and, thus, (68) holds iff

$$\mathit{sum}(t_{set})^I = \mathit{weight}(d^*_{tuple})^I + \sum\{\mathit{weight}(d^*)^I \mid d \in d_{set} \text{ and } \mathit{weight}(d^*)^I \neq 0\}$$

iff

$$sum(t_{set})^I = weight(d^*_{tuple})^I + sum(d^*_{set})^I \tag{69}$$

Furthermore, since I is a standard interpretation that satisfies condition 19 and d_{tuple} is a non-zero weight tuple that belongs to t^I_{set}, it follows that I satisfies $d^*_{tuple} \in t_{set} \wedge weight(d^*_{tuple}) \neq 0$. Hence, it is enough to show that I satisfies formula

$$\exists N(sum(rem(t_{set}, d^*_{tuple})) = N \wedge sum(t_{set}) = N + weight(d^*_{tuple})) \tag{70}$$

iff (69) holds. Note also that, since I satisfies condition 17, it follows that

$$rem(t_{set}, d^*_{tuple})^I = t^I_{set} \setminus \{d_{tuple}\} = d_{set}$$

and, thus, formula (70) can be rewritten as

$$\exists N(sum(d^*_{set}) = N \wedge sum(t_{set}) = N + weight(d^*_{tuple})) \tag{71}$$

Therefore, we just need to show that I satisfies formula (71) iff (69) holds. It is easy to see that (71) implies (69). Note also that $sum(d^*_{set})^I$ is an integer and, thus, (69) also implies (71). □

Lemma 21. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation and sentence* (24)*. Let t_{set} be a term of sort s_{set} such that t^I_{set} contains finitely many tuples with non-zero weights. Then, I satisfies formula*

$$\forall T\big(T \in d^*_{set} \wedge weight(T) \neq 0 \rightarrow \exists N\big(sum(rem(d^*_{set}, T)) = N \wedge sum(d^*_{set}) = N + weight(T)\big)\big) \tag{72}$$

for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set} iff

$$sum(d^*_{set})^I \;=\; \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in d_{set} \text{ and } weight(d^*_{tuple})^I \neq 0\} \tag{73}$$

for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set}.

Proof. The proof follows by induction assuming that the lemma statement holds for all domain elements that have strictly fewer tuples with non-zero weights.

Base case. In the case where t^I_{set} does not have any tuple with non-zero weight, since I is a standard interpretation, it follows that I satisfies (72). This also implies that I satisfies

$$\forall T\,(T \in t_{set} \rightarrow weight(T) = 0)$$

and, thus, $sum(d^*_{set})^I = 0$ holds for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set}. Hence, the lemma statement holds.

The *induction step* follows directly from Lemma 20. □

Lemma 22. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation and sentence* (24)*. Let t_{set} be a term of sort s_{set} such that t^I_{set} contains finitely many tuples with non-zero weights. Then, I satisfies formula*

$$\forall T\big(T \in d^*_{set} \rightarrow \exists N\big(sum(rem(d^*_{set}, T)) = N \wedge sum(d^*_{set}) = N + weight(T)\big)\big) \tag{74}$$

for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set} iff

$$sum(d^*_{set})^I \;=\; \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in d_{set} \text{ and } weight(d^*_{tuple})^I \neq 0\} \tag{75}$$

for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set}.

Proof. Right-to-left. From Lemma 21, it follows (75) holds for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set} iff I satisfies (72) for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set}. Furthermore, (74) entails (72) in first order logic.

Left-to-right. Note (74) is equivalent to the conjunction of (72) and

$$\forall T\big(T \in d^*_{set} \wedge weight(T) = 0 \rightarrow \exists N\big(sum(rem(d^*_{set}, T)) = N \wedge sum(d^*_{set}) = N + weight(T)\big)\big) \quad (76)$$

Assume that (75) holds for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set}. Then, from Lemma 21, interpretation I satisfies (72) for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set}. Furthermore, for standard interpretations, (76) is equivalent to

$$\forall T\big(T \in d^*_{set} \wedge weight(T) = 0 \rightarrow \exists N\,(sum(rem(d^*_{set}, T)) = N \wedge sum(d^*_{set}) = N)\big)$$

Pick any domain element d_{tuple} of sort s_{tuple} s.t. I satisfies $d_{tuple} \in d^*_{set} \wedge weight(d_{tuple}) = 0$. We need to show

$$sum(d^*_{set})^I = sum(rem(d_{set}, d_{tuple}))^I \quad (77)$$

which follows from the fact that (75) holds for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set} and that d_{set} and $rem(d_{set}, d_{tuple})^I$ have the same tuples with non-zero weights. □

Lemma 23. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation and sentence* (24)*. Then, I satisfies sentence* (25) *iff*

$$sum(e^*_{set})^I = \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in e_{set},\ weight(d^*_{tuple})^I \neq 0\} \quad (78)$$

for every domain element e_{set} of sort s_{set} that contains a finite number of non-zero weight tuples.

Proof. Left-to-right. Assume first that I satisfies sentence (25). Pick any domain element e_{set} of sort s_{set} that contains a finite number of non-zero weight tuples. Then, I satisfies $FiniteWeight(d^*_{set})$ for every subset d_{set} of e_{set} (Lemma 19) and, since it satisfies (25), it follows that I satisfies $FiniteSum(d^*_{set})$ for every subset d_{set} of e_{set}. Since I also satisfies (24), we get that it satisfies (74) for every domain element d_{set} of sort s_{set} which is a subset of e_{set}. From Lemma 22, this implies that (78) holds.

Right-to-left. Pick any domain element e_{set} of sort s_{set} and assume that I satisfies $FiniteWeight(e^*_{set})$. From Lemma 19, this implies that e_{set} contains a finite number of tuples with non-zero weights and, thus, so does any subset d_{set} of e_{set}. By hypothesis, this implies that

$$sum(d^*_{set})^I \ = \ \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in d_{set} \text{ and } weight(d^*_{tuple})^I \neq 0\}$$

holds for every subset d_{set} of e_{set}. From Lemma 22, this implies that I satisfies

$$\forall T\big(T \in d^*_{set} \rightarrow \exists N\big(sum(rem(d^*_{set}, T)) = N \wedge sum(d^*_{set}) = N + weight(T)\big)\big)$$

for every subset d_{set} of e_{set}. In particular, this implies that I satisfies

$$d^*_{tuple} \in e^*_{set} \rightarrow \exists N\big(sum(rem(e^*_{set}, T)) = N \wedge sum(e^*_{set}) = N + weight(T)\big)$$

Therefore, I satisfies (25). □

Proof of Proposition 4. *Left-to-right*. Assume first that I satisfies condition 7 and we will show that I satisfies sentences (24-26). Since I is a standard interpretation, $sum(t_{set})^I = 0$ holds for any t_{set} without non-zero weight tuples and, thus, I satisfies (24). Furthermore, from Lemma 23, I satisfies (25). Finally, to show that I satisfies (26), pick any domain element d_{set} of sort s_{set} and assume that I does not satisfy $FiniteWeight(d^*_{set})$. From Lemma 19, this implies that d_{set} contains an infinite number of tuples with non-zero weights and, since I satisfies condition 7, it follows that I satisfies $sum(d^*_{set}) = 0$. Therefore, I satisfies (26).

Right-to-left. Assume that I satisfies sentences (24-26) and we will show that it satisfies condition 7. Pick any term t_{set} of sort s_{set}. If t^I_{set} contains no tuples with non-zero weights, (24) implies that $sum(t_{set}) = 0$. Similarly, if t^I_{set} contains an infinite number of tuples with non-zero weights, (26) implies that $sum(t_{set}) = 0$. It only remains to be shown that, if t^I_{set} contains a finite number of tuples with non-zero weights, then

$$sum(t_{set})^I \;=\; \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in t^I_{set} \text{ and } weight(d^*_{tuple})^I \neq 0\} \tag{79}$$

which follows from Lemmas 19 and 23 and the fact that I satisfies (25). □

A.5 Proof of Proposition 5

Lemma 24. *Let I be a standard interpretation and t_{set} be a term of sort s_{set}. Then, I satisfies formula $FinitePositive(t_{set})$ iff set $\{d \in t^I_{set} \mid I \models weight(d^*) > 0\}$ is finite, that is, iff there is a bijection between this set and a set of natural numbers of form $\{i \in \mathbb{N} \mid i \leq n\}$ for some natural number n.*

Proof. The proof is analogous to the proof of Lemma 19. □

Lemma 25. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation. Let t_{set} be a term of sort s_{set} such that t^I_{set} contains finitely many and at least one tuple with positive weight and such that I satisfies*

$$sum^+(d^*_{set})^I \;=\; \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in d_{set} \text{ and } weight(d^*_{tuple})^I > 0\} \tag{80}$$

for every domain element d_{set} of sort s_{set} which has strictly less positive tuples than t^I_{set}. Then, I satisfies formula

$$\begin{aligned}\forall T \big(T \in t_{set} \wedge weight(T) > 0 \rightarrow \\ \exists N \big(sum^+(rem(t_{set}, T)) = N \wedge sum^+(t_{set}) = N + weight(T)\big)\big)\end{aligned} \tag{81}$$

iff

$$sum^+(t_{set})^I \;=\; \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in t^I_{set} \text{ and } weight(d^*_{tuple})^I > 0\} \tag{82}$$

Proof. The proof is analogous to the proof of Lemma 20. □

Lemma 26. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation and sentence* (27)*. Let t_{set} be a term of sort s_{set} such that t^I_{set} contains finitely many tuples with positive weights. Then, I satisfies formula*

$$\begin{aligned}\forall T \big(T \in d^*_{set} \wedge weight(T) > 0 \rightarrow \\ \exists N \big(sum^+(rem(d^*_{set}, T)) = N \wedge sum^+(d^*_{set}) = N + weight(T)\big)\big)\end{aligned} \tag{83}$$

for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set} iff

$$sum^+(d^*_{set})^I = \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in d_{set},\ weight(d^*_{tuple})^I > 0\} \tag{84}$$

for every domain element d_{set} of sort s_{set} which is a subset of t^I_{set}.

Proof. The proof is analogous to the proof of Lemma 21. □

Lemma 27. *Let I be a standard interpretation that satisfies conditions 5, 17 and 19 for being an agg-interpretation and sentence* (27)*. Then, I satisfies sentence* (28) *iff*

$$sum^+(e^*_{set})^I = \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in e_{set} \text{ and } weight(d^*_{tuple})^I > 0\} \tag{85}$$

for every domain element e_{set} of sort s_{set} that contains a finite number of positive weight tuples.

Proof. The result follows from Lemma 26 in a similar way that Lemma 23 follows from Lemma 22. □

Proof of Proposition 5. *Left-to-right*. Assume first that I satisfies condition 8 and we will show that I satisfies sentences (27-29). Since I is a standard interpretation, $sum^+(t_{set})^I = 0$ holds for any t_{set} without positive weight tuples and, thus, I satisfies (27). Furthermore, from Lemma 27, I satisfies (28). Finally, to show that I satisfies (29), pick any domain element d_{set} of sort s_{set} and assume that I does not satisfy $FinitePositive(d^*_{set})$. From Lemma 24, this implies that d_{set} contains an infinite number of tuples with positive weights and, since I satisfies condition 8, it follows that I satisfies $sum^+(d^*_{set}) = sup$. Therefore, I satisfies (29).

Right-to-left. Assume that I satisfies sentences (27-29) and we will show that it satisfies condition 8. Pick any term t_{set} of sort s_{set}. If t^I_{set} contains no tuples with positive weights, (27) implies that $sum^+(t_{set}) = 0$. Similarly, if t^I_{set} contains an infinite number of tuples with positive weights, (29) implies that $sum^+(t_{set}) = sup$. It only remains to be shown that, if t^I_{set} contains a finite number of tuples with positive weights, then

$$sum^+(t_{set})^I \ = \ \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in t^I_{set} \text{ and } weight(d^*_{tuple})^I > 0\} \tag{86}$$

which follows from Lemmas 24 and 27 and the fact that I satisfies (28). □

A.6 Proof of Propositions 6-7

Lemma 28. *Let I be an interpretation that satisfies all conditions for being an agg-interpretation except conditions 6-8 and 10, and let t_{set} be a term of sort s_{set}. If $I \models Finite(t_{set})$ then $I \models FiniteMin(t_{set})$.*

Proof. By Lemma 15, t^I_{set} is a finite set. Pick any domain element d_{tuple} of sort s_{tuple}. *Case 1.* d_{tuple} does not belong to t^I_{set}. Then, the antecedent of $FiniteMin(t_{set})$ is false. *Case 2.* d_{tuple} belongs to t^I_{set}. Due to the definition of $\widehat{\texttt{min}}$ for non-empty finite sets, $\widehat{\texttt{min}}(t^I_{set})$ is the least element of the set

$$\{X_1 \mid \langle X_1, \ldots, X_k \rangle \in t^I_{set}\}. \tag{87}$$

Assume that $I \models \neg\exists T'(T \in t_{set} \wedge first(T') < first(d^*_{tuple}))$. Otherwise, the antecedent of $FiniteMin(t_{set})$ does not hold. By conditions 13, 16, and 18 of agg-interpretations, there is no tuple $\langle d_1, \ldots, d_k \rangle \in t^I_{set}$ such that $d_1 < first(d^*_{tuple})^I$. This implies that $first(d^*_{tuple})^I$ is the least element of the set (87).

Thus, $\widehat{\texttt{min}}(t_{set}^I)$ is $first(d_{tuple}^*)^I$. By condition 9, $\widehat{\texttt{min}}(t_{set}^I)$ is $min(t_{set})^I$ and consequently, $min(t_{set})^I$ is $first(d_{tuple}^*)^I$. Hence, I satisfies

$$\forall T \big(T \in t_{set} \wedge \neg \exists T' \big(T' \in t_{set} \wedge first(T') < first(T)\big) \rightarrow min(t_{set}) = first(T)\big)$$

that is, $I \models FiniteMin(t_{set})$. □

Lemma 29. *Let I be an interpretation that satisfies all conditions for being an agg-interpretation except conditions 6-10, and let t_{set} be a term of sort s_{set}. Further assume that*

$$I \models \forall S\,(Finite(S) \rightarrow FiniteMin(S))$$

and that t_{set}^I is finite and non-empty. Then I satisfies the condition that $min(t_{set})^I$ is the least element of the set (87)*.*

Proof. Since t_{set}^I is finite, by Lemma 15 it follows that $I \models Finite(t_{set})$. Thus, $I \models FiniteMin(t_{set})$ as well. That is, I satisfies

$$\forall T \big(T \in t_{set} \wedge \neg \exists T' \big(T' \in t_{set} \wedge first(T') < first(T)\big) \rightarrow min(t_{set}) = first(T)\big)$$

By assumption and condition 4 of agg-interpretations, it follows that t_{set}^I contains at least one tuple and thus, there exists a tuple $d_{tuple} = \langle d_1, \ldots, d_k \rangle$ in t_{set}^I such that d_1 is the least element of the set (87). Since d_{tuple} belongs to set t_{set}^I and $I \models FiniteMin(t_{set})$, it follows that I satisfies

$$d_{tuple}^* \in t_{set} \wedge \neg \exists T' \big(T' \in t_{set} \wedge first(T') < d_1^*\big) \rightarrow min(t_{set}) = d_1^*$$

For the sake of contradiction, suppose that I satisfies

$$\exists T' \big(T' \in t_{set} \wedge first(T') < d_1^*\big)$$

Then, there is $d'_{tuple} = \langle e_1, \ldots, e_k \rangle$ such that I satisfies

$$d_{tuple}'^* \in t_{set} \wedge e_1^* < d_1^*$$

This implies that e_1 belongs to set (87) and that $e_1 < d_1$, which is a contradiction with the fact that d_1 is the least element of the set (87). Therefore, I satisfies

$$\neg \exists T' \big(T' \in t_{set} \wedge first(T') < first(d_{tuple}^*)\big)$$

and, thus, I satisfies that $min(t_{set}) = d_1^*$. From this, it follows that $min(t_{set})^I$ is the least element of the set (87). □

Proof of Proposition 6. *Left-to-right*. Assume I satisfies condition 9. Then, for any domain element d_{set}, if d_{set} is empty, $\widehat{\texttt{min}}(d_{set})$ is sup. Since I is a standard interpretation, $\overline{\emptyset}^I = \emptyset$. Thus,

$$min(d_{set}^*)^I = min(\overline{\emptyset})^I = \widehat{\texttt{min}}(\emptyset) = sup\,.$$

Now choose an arbitrary term d_{set} of sort s_{set} and assume $I \models Finite(d_{set}^*)$. By Lemma 28, $I \models FiniteMin(d_{set}^*)$. Choose an arbitrary domain element d_{set} of sort s_{set} such that $I \not\models Finite(d_{set}^*)$. By

Lemma 15, d_{set} is *not* a finite set. Due to the definition of $\widehat{\texttt{min}}$ for infinite sets, $\widehat{\texttt{min}}(d_{set})$ is *inf*. Thus,

$$min(d^*_{set})^I = min^I(d_{set}) = \widehat{\texttt{min}}(d_{set}) = inf\,.$$

Therefore, I satisfies sentences (30-32).

Right-to-left. Assume I satisfies sentences (30-32). Since $\overline{\emptyset}^I = \emptyset$ and I satisfies sentence (30), $min^I(\emptyset)$ is *sup*. This satisfies the first condition of $\widehat{\texttt{min}}$. Now choose a domain element d_{set} that is finite and non-empty. By Lemma 29, $min(d^*_{set})^I$ is the least element of the set of elements consisting of the first members of elements from d_{set}. This satisfies the second condition of $\widehat{\texttt{min}}$. Finally, choose a domain element d_{set} that is infinite. Then, by Lemma 15, $I \not\models Finite(d_{set})$. Since I satisfies sentence (32), $I \models min(d^*_{set}) = inf$. Therefore, $min^I(d_{set})$ is *inf*. This satisfies the last condition of $\widehat{\texttt{min}}$. Therefore, I satisfies condition 9. □

Lemma 30. *Let I be an interpretation that satisfies all conditions for being an agg-interpretation except conditions 6-9, and let t_{set} be a term of sort s_{set}. In this case, if $I \models Finite(t_{set})$ then $I \models FiniteMax(t_{set})$.*

Proof. The proof is symmetric to the proof of Lemma 28. □

Lemma 31. *Let I be an interpretation that satisfies all conditions for being an agg-interpretation except conditions 6-10, and let t_{set} be a term of sort s_{set}. Further assume that*

$$I \models \forall S\,(Finite(S) \rightarrow FiniteMax(S))$$

and that t^I_{set} is finite and non-empty. Then I satisfies the condition that $max(t_{set})^I$ is the greatest element of the set (87)*.*

Proof. The proof is symmetric to the proof of Lemma 29. □

Proof of Proposition 7. *Left-to-right*. Assume I satisfies condition 10. By assumption, for any domain element d_{set}, if d_{set} is empty, $\widehat{\texttt{max}}(d_{set})$ is *inf*. Since I is a standard interpretation, $\overline{\emptyset}^I = \emptyset$. Thus,

$$max(d^*_{set})^I = max(\overline{\emptyset})^I = \widehat{\texttt{max}}(\emptyset) = inf\,.$$

Now choose an arbitrary term d_{set} of sort s_{set} and assume $I \models Finite(d^*_{set})$. By Lemma 30, $I \models FiniteMax(d^*_{set})$. Choose an arbitrary domain element d_{set} of sort s_{set} such that $I \not\models Finite(d^*_{set})$. By Lemma 15, d_{set} is *not* a finite set. Due to the definition of $\widehat{\texttt{max}}$ for infinite sets, $\widehat{\texttt{max}}(d_{set})$ is *sup*. Thus,

$$max(d^*_{set})^I = max^I(d_{set}) = \widehat{\texttt{max}}(d_{set}) = sup\,.$$

Therefore, I satisfies sentences 30-32.

Right-to-left. Assume I satisfies sentences 33-35. Choose an arbitrary domain element d_{set} such that d_{set} is empty. Recall that $\overline{\emptyset}^I = \emptyset$. Since I satisfies sentence 33, $max^I(\emptyset)$ is *inf*. This satisfies the first condition of $\widehat{\texttt{max}}$.

Now choose an arbitrary domain element d_{set} such that d_{set} is finite and non-empty. By Lemma 31, $max(d^*_{set})^I$ is the greatest element of the set of elements consisting of the first elements of d_{set}. This satisfies the second condition of $\widehat{\texttt{max}}$.

Now choose an arbitrary domain element d_{set} such that d_{set} is infinite. Then, by Lemma 15, $I \not\models Finite(d_{set})$. Since I satisfies sentence 35, $I \models max(d^*_{set}) = sup$. Therefore, $min^I(d_{set})$ is *sup*. This satisfies the last condition of $\widehat{\texttt{max}}$. Therefore, I satisfies condition 10. □

A.7 Proof of Theorem 3

Proof of Theorem 3. A set M of ground atoms is an answer set of Π
iff there is an agg-interpretation I that is a model of $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$
and $M = Ans(I)$
iff there is a standard interpretation I that is a model of $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ satisfying conditions 5-10 and 17-19, and $M = Ans(I)$
iff there is standard interpretation I that is a model of $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ satisfying conditions 5 and 17-19, and sentences (21-35), and $M = Ans(I)$ (Propositions 3-7)
iff there is a standard interpretation I that is a model of $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ satisfying sentences (16-35) and $M = Ans(I)$ (Propositions 1 and 2)
□

A.8 Proof of Theorem 8

The lemmas of this section all make the following assumptions. Let E be an aggregate element. Let I be a standard interpretation that satisfies conditions 5, 17-19 for being an agg-interpretation and sentences (21,24,27,30,33) and (38-42). Let J be an agg-interpretation that agrees with I in the interpretation of all symbols but the function symbols *sum*, *count*, sum^+, *min*, and *max*. Note that, since I and J agree on the interpretation of all symbols except the aggregate symbols, $set_{|E|}(\mathbf{t})^I = set_{|E|}(\mathbf{t})^J$ for any list $\mathbf{t}$ of terms of sort s_{prg} of the correct length.

Lemma 32. *If $set_{|E|}(\mathbf{t})^I$ contains finitely many tuples, then $count(set_{|E|}(\mathbf{t}))^I = count(set_{|E|}(\mathbf{t}))^J$ holds for any list $\mathbf{t}$ of terms of sort s_{prg} of the correct length.*

Proof. Since I satisfies sentence (38), it follows that I satisfies $FiniteCount(d_{set})$ for every domain element d_{set} of sort s_{set} that is a subset of $set_{|E|}(\mathbf{t})$. Equivalently, I satisfies sentence (60) for every $d_{set} \subseteq set_{|E|}(\mathbf{t})$. From Lemma 17 and the fact that J is an agg-interpretation, this implies

$$count(set_{|E|}(\mathbf{t}))^I = |set_{|E|}(\mathbf{t})^I| = |set_{|E|}(\mathbf{t})^J| = count(set_{|E|}(\mathbf{t}))^J$$

□

Lemma 33. *If $set_{|E|}(\mathbf{t})^I$ contains finitely many tuples with non-zero weights, then $sum(set_{|E|}(\mathbf{t}))^I = sum(set_{|E|}(\mathbf{t}))^J$ holds for any list $\mathbf{t}$ of terms of sort s_{prg} of the correct length.*

Proof. Since I satisfies sentence (39), it follows that I satisfies $FiniteSum(d_{set})$ for every domain element d_{set} of sort s_{set} that is a subset of $set_{|E|}(\mathbf{t})$. From Lemma 22 and the fact that J is an agg-interpretation, this implies

$$\begin{aligned} sum(set_{|E|}(\mathbf{t}))^I &= \sum\{weight(d^*_{tuple})^I \mid d_{tuple} \in set_{|E|}(\mathbf{t})^I \text{ and } weight(d^*_{tuple})^I \neq 0\} \\ &= \sum\{weight(d^*_{tuple})^J \mid d_{tuple} \in set_{|E|}(\mathbf{t})^J \text{ and } weight(d^*_{tuple})^J \neq 0\} \\ &= sum(set_{|E|}(\mathbf{t}))^J \end{aligned}$$

□

Lemma 34. *If $set_{|E|}(\mathbf{t})^I$ contains finitely many tuples with positive weights, then $sum^+(set_{|E|}(\mathbf{t}))^I = sum^+(set_{|E|}(\mathbf{t}))^J$ holds for any list $\mathbf{t}$ of terms of sort s_{prg} of the correct length.*

Proof. Since I satisfies sentence (40), it follows that I satisfies $\mathit{FinitePositiveSum}(d_{set})$ for every domain element d_{set} of sort s_{set} that is a subset of $\mathit{set}_{|E|}(\mathbf{t})$. From Lemma 26 and the fact that J is an agg-interpretation, this implies

$$\begin{aligned}\mathit{sum}^{+}(\mathit{set}_{|E|}(\mathbf{t}))^I &= \sum\{\mathit{weight}(d^*_{tuple})^I \mid d_{tuple} \in \mathit{set}_{|E|}(\mathbf{t})^I \text{ and } \mathit{weight}(d^*_{tuple})^I > 0\} \\ &= \sum\{\mathit{weight}(d^*_{tuple})^J \mid d_{tuple} \in \mathit{set}_{|E|}(\mathbf{t})^J \text{ and } \mathit{weight}(d^*_{tuple})^J > 0\} \\ &= \mathit{sum}^{+}(\mathit{set}_{|E|}(\mathbf{t}))^J\end{aligned}$$

□

Lemma 35. *If $\mathit{set}_{|E|}(\mathbf{t})^I$ contains finitely many tuples, then $\mathit{min}(\mathit{set}_{|E|}(\mathbf{t}))^I = \mathit{min}(\mathit{set}_{|E|}(\mathbf{t}))^J$ holds for any list $\mathbf{t}$ of terms of sort s_{prg} of the correct length.*

Proof. Case 1: $\mathit{set}_{|E|}(\mathbf{t})^I$ contains at least one tuple. Since I satisfies sentence (41), it follows that I satisfies $\mathit{FiniteMin}(\mathit{set}_{|E|}(\mathbf{t}))$. By assumption, $\mathit{set}_{|E|}(\mathbf{t})^I$ contains at least one tuple, therefore, there exists a tuple d_{tuple} in $\mathit{set}_{|E|}(\mathbf{t})^I$ such that d_1 is the least element of the set

$$\{X_1 \mid \langle X_1, \ldots, X_k\rangle \in \mathit{set}_{|E|}(\mathbf{t})^I\} \tag{88}$$

and I satisfies

$$d_{tuple} \in \mathit{set}_{|E|}(\mathbf{t}) \wedge \neg\exists T'(T' \in \mathit{set}_{|E|}(\mathbf{t}) \wedge \mathit{first}(T') < d^*_1) \rightarrow \mathit{min}(\mathit{set}_{|E|}(\mathbf{t})) = d^*_1$$

For the sake of contradiction, suppose that I satisfies

$$\exists T'\big(T' \in \mathit{set}_{|E|}(\mathbf{t}) \wedge \mathit{first}(T') < d^*_1\big)$$

Then, there exists $c_{tuple} = \langle e_1, \ldots, e_k\rangle$ such that I satisfies

$$c^*_{tuple} \in \mathit{set}_{|E|}(\mathbf{t}) \wedge e^*_1 < d^*_1$$

This implies that e_1 belongs to set (88) and that $e_1 < d_1$, which is a contradiction with the fact that d_1 is the least element of the set (88). Therefore, I satisfies

$$\neg\exists T'\big(T' \in \mathit{set}_{|E|}(\mathbf{t}) \wedge \mathit{first}(T') < \mathit{first}(d^*_{tuple})\big)$$

and, thus, I satisfies that $\mathit{min}(\mathit{set}_{|E|}(\mathbf{t})) = d^*_1$. From this, it follows that $\mathit{min}(\mathit{set}_{|E|}(\mathbf{t}))^I$ is the least element of the set (88). Since I and J agree on the interpretation of all symbols except the aggregate function symbols,

$$\{X_1 \mid \langle X_1, \ldots, X_k\rangle \in \mathit{set}_{|E|}(\mathbf{t})^I\} = \{X_1 \mid \langle X_1, \ldots, X_k\rangle \in \mathit{set}_{|E|}(\mathbf{t})^J\} \tag{89}$$

and since J is an agg-interpretation, $\mathit{min}(\mathit{set}_{|E|}(\mathbf{t}))^J$ is also the least element of this set. Consequently, $\mathit{set}_{|E|}(\mathbf{t})^I = \mathit{min}(\mathit{set}_{|E|}(\mathbf{t}))^J$

Case 2: $\mathit{set}_{|E|}(\mathbf{t})^I$ contains no tuples. Since I is a standard interpretation, $\mathit{set}_{|E|}(\mathbf{t})^I = \emptyset$. By assumption, I satisfies sentence 30 and J is an agg-interpretation, thus,

$$\mathit{min}(\mathit{set}_{|E|}(\mathbf{t}))^I = \mathit{sup} = \mathit{min}(\mathit{set}_{|E|}(\mathbf{t}))^J$$

□

Lemma 36. *If $set_{|E|}(\mathbf{t})^I$ contains finitely many tuples, then $max(set_{|E|}(\mathbf{t}))^I = max(set_{|E|}(\mathbf{t}))^J$ holds for any list $\mathbf{t}$ of terms of sort s_{prg} of the correct length.*

Proof. The proof is symmetric to the proof of Lemma 35. □

Proof of Theorem 8. *Left-to-right.* Assume that M is an answer set of some program Π. Then, from Theorem 3, there is some standard model I of $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ that satisfies all sentences of forms (16-35) and $M = Ans(I)$. Hence, it only remains to be shown that I satisfies all sentences of forms (38-42). Pick any aggregate element E, any list $\mathbf{d}$ of domain elements of sort s_{prg} and any domain element d_{set} of sort s_{set} and assume that I satisfies $Subset(d^*_{set}, set_{|E|}(\mathbf{d}^*))$.
Now we will show that I satisfies
$FiniteCount(d^*_{set})$, $FiniteSum(d^*_{set})$, $FinitePositiveSum(d^*_{set})$,
$FiniteMin(set_{|E|}(\mathbf{d}^*))$, and $FiniteMax(set_{|E|}(\mathbf{d}^*))$.
Note that I satisfies the following sentences:

$Finite(d^*_{set}) \rightarrow FiniteCount(d^*_{set})$ (since $I \models$ (22))
$FiniteWeight(d^*_{set}) \rightarrow FiniteSum(d^*_{set})$ (since $I \models$ (25))
$FinitePositive(d^*_{set}) \rightarrow FinitePositiveSum(d^*_{set})$ (since $I \models$ (28))
$Finite(set_{|E|}(\mathbf{d}^*)) \rightarrow FiniteMin(set_{|E|}(\mathbf{d}^*))$ (since $I \models$ (31))
$Finite(set_{|E|}(\mathbf{d}^*)) \rightarrow FiniteMax(set_{|E|}(\mathbf{d}^*))$ (since $I \models$ (34))

Furthermore, since Π has finite aggregates, it follows that I has finite aggregates and, thus, that $set_{|E|}(\mathbf{d}^*)^I$ is finite. Since d_{set} is a subset of $set_{|E|}(\mathbf{d}^*)^I$, it is also finite. From Lemma 15, I satisfies $Finite(set_{|E|}(\mathbf{d}^*))$, and so I must also satisfy $FiniteMin(set_{|E|}(\mathbf{d}^*))$, and $FiniteMax(set_{|E|}(\mathbf{d}^*))$. From Lemma 15, this implies that I satisfies $Finite(d^*_{set})$, and so I must also satisfy $FiniteCount(d^*_{set})$, $FiniteMin(set_{|E|}(\mathbf{d}^*))$ and $FiniteMax(set_{|E|}(\mathbf{d}^*))$. From Lemma 19, the above also implies that I satisfies $FiniteWeight(d^*_{set})$, and so I must also satisfy $FiniteSum(d^*_{set})$. From Lemma 24, this also implies that I satisfies $FinitePositive(d^*_{set})$, and consequently I must also satisfy $FinitePositiveSum(d^*_{set})$. Therefore, I satisfies all sentences of forms (38-42).

Right-to-left. Assume that I is a standard model of $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ that satisfies all sentences of form (16-21,24,27,30,33,38-42) and $M = Ans(I)$. From Proposition 2, interpretation I satisfies conditions 5, 17-19 for being an agg-interpretation. Let J be an agg-interpretation that agrees with I in the interpretation of all symbols but *count*, *sum*, sum^+, *min*, and *max*. Further assume that $count^J$, sum^J, sum^{+J}, min^J, and max^J are defined according to conditions 6, 7, 8, 9, and 10, respectively. Then, J is an agg-interpretation and $M = Ans(I) = Ans(J)$. It only remains to be shown that J satisfies $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$. Since I and J only differ in the interpretation of the aggregate function symbols, and in $\mathrm{SM}_{\mathbf{p}}[\kappa\Pi]$ these function symbols only occur when applied to terms of form $set_{|E|}(\mathbf{t})$, it is enough to show that the sentences

$$count(set_{|E|}(\mathbf{t}))^I = count(set_{|E|}(\mathbf{t}))^J$$
$$sum(set_{|E|}(\mathbf{t}))^I = sum(set_{|E|}(\mathbf{t}))^J$$
$$sum^+(set_{|E|}(\mathbf{t}))^I = sum^+(set_{|E|}(\mathbf{t}))^J$$
$$min(set_{|E|}(\mathbf{t}))^I = min(set_{|E|}(\mathbf{t}))^J$$
$$max(set_{|E|}(\mathbf{t}))^I = max(set_{|E|}(\mathbf{t}))^J$$

hold for every aggregate element E and list $\mathbf{t}$ of terms of sort s_{prg} of the correct length. Since I satisfies (16), from Proposition 1 it follows that it satisfies condition 5 for being an agg-interpretation.

This also implies that

$$set_{|E|}(\mathbf{t})^I = set_{|E|}(\mathbf{t})^J$$

Furthermore, since Π has finite aggregates, these two sets are finite. In addition, since I satisfies (17-20), from Proposition 2, it follows that it satisfies conditions 17-19. Therefore, the result follows from Lemmas 32-36. □

References

Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., & Wanko, P. (2021). Train scheduling with hybrid answer set programming. *Theory and Practice of Logic Programming*, *21*(3), 317–347.

Adrian, W. T., Alviano, M., Calimeri, F., Cuteri, B., Dodaro, C., Faber, W., Fuscà, D., Leone, N., Manna, M., Perri, S., Ricca, F., Veltri, P., & Zangari, J. (2018). The ASP system DLV: Advancements and applications. *KI - Künstliche Intelligenz*, *32*, 177 – 179.

Aker, E., Erdogan, A., Erdem, E., & Patoglu, V. (2011). Causal reasoning for planning and coordination of multiple housekeeping robots. In Delgrande, J. P., & Faber, W. (Eds.), *Logic Programming and Nonmonotonic Reasoning*, pp. 311–316, Berlin, Heidelberg. Springer Berlin Heidelberg.

Asuncion, V., Chen, Y., Zhang, Y., & Zhou, Y. (2015). Ordered completion for logic programs with aggregates. *Artificial Intelligence*, *224*, 72–102.

Asuncion, V., Lin, F., Zhang, Y., & Zhou, Y. (2012). Ordered completion for first-order logic programs on finite structures. *Artificial Intelligence*, *177-179*(1-2), 1–24.

Balduccini, M. (2011). Industrial-size scheduling with ASP+cp. In *Logic Programming and Nonmonotonic Reasoning - 11th International Conference (LPNMR)*, Vol. 6645 of *Lecture Notes in Computer Science*, pp. 284–296. Springer-Verlag.

Balduccini, M., & Gelfond, M. (2005). Model-based reasoning for complex flight systems. In *Proceedings of Infotech@Aerospace (American Institute of Aeronautics and Astronautics)*.

Balduccini, M., Gelfond, M., Nogueira, M., Watson, R., & Barry, M. (2001). An A-Prolog decision support system for the Space Shuttle. In *Working Notes of the AAAI Spring Symposium on Answer Set Programming*.

Bartholomew, M., Lee, J., & Meng, Y. (2011). First-order extension of the flp stable model semantics via modified circumscription.. In Walsh, T. (Ed.), *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 724–730. IJCAI/AAAI Press.

Cabalar, P. (2011). Functional answer set programming. *Theory and Practice of Logic Programming*, *11*(2-3), 203–233.

Cabalar, P., Fandinno, J., Fariñas del Cerro, L., & Pearce, D. (2018). Functional ASP with intensional sets: Application to Gelfond-Zhang aggregates. *Theory and Practice of Logic Programming*, *18*(3-4), 390–405.

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

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

Calogero G. Zarba (2006). Chapter 1 - Many-Sorted Logic. `https://www.react.uni-saarland.de/teaching/decision-procedures-verification-06/ch01.pdf`. Online; accessed 13 November 2020.

Cao Tran, S., Pontelli, E., Balduccini, M., & Schaub, T. (2023). Answer set planning: A survey. *Theory and Practice of Logic Programming*, *23*(1), 226–298.

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

Faber, W., Pfeifer, G., & Leone, N. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, *175*(1), 278–298.

Fandinno, J., Hansen, Z., & Lierler, Y. (2022a). Arguing correctness of ASP programs with aggregates.. Vol. 13416 of *Lecture Notes in Computer Science*, pp. 190–202. Springer.

Fandinno, J., Hansen, Z., & Lierler, Y. (2022b). Axiomatization of aggregates in answer set programming. In *Proceedings of the Thirty-sixth National Conference on Artificial Intelligence (AAAI'22)*, pp. 5634–5641. AAAI Press.

Fandinno, J., Lifschitz, V., Lühne, P., & Schaub, T. (2020). Verifying tight logic programs with anthem and vampire. *Theory and Practice of Logic Programming*, *20*(5), 735–750.

Fandinno, J., Hansen, Z., Lierler, Y., Lifschitz, V., & Temple, N. (2023). External behavior of a logic program and verification of refactoring..

Fandinno, J., & Lifschitz, V. (2023). Verification of locally tight programs..

Ferraris, P. (2011). Logic programs with propositional connectives and aggregates. *ACM Transactions on Computational Logic*, *12*(4), 25:1–25:40.

Ferraris, P., Lee, J., & Lifschitz, V. (2011). Stable models and circumscription. *Artificial Intelligence*, *175*(1), 236–263.

Ferraris, P., & Lifschitz, V. (2010). On the stable model semantics of first-order formulas with aggregates. In *Proceedings of the 2010 Workshop on Nonmonotonic Reasoning*.

Gebser, M., Harrison, A., Kaminski, R., Lifschitz, V., & Schaub, T. (2015). Abstract Gringo. *Theory and Practice of Logic Programming*, *15*(4-5), 449–463.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2019). Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming*, *19*(1), 27–82.

Gebser, M., Kaminski, R., & Schaub, T. (2011). aspcud: A linux package configuration tool based on answer set programming.. In *Second International Workshop on Logics for Component Configuration (LoCoCo)*.

Gebser, M., Obermeier, P., Otto, T., Schaub, T., Sabuncu, O., Nguyen, V., & Son, T. C. (2018). Experimenting with robotic intra-logistics domains. *Theory and Practice of Logic Programming*, *18*(3-4), 502–519.

Gelfond, M., & Zhang, Y. (2014). Vicious circle principle and logic programs with aggregates. *Theory and Practice of Logic Programming*, *14*(4-5), 587–601.

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

Harrison, A., & Lifschitz, V. (2018). Relating two dialects of answer set programming. In Fermé, E., & Villata, S. (Eds.), *Proceedings of the Seventeenth International Workshop on Nonmonotonic Reasoning (NMR'18)*, pp. 99–108.

Harrison, A., & Lifschitz, V. (2019). Relating two dialects of answer set programming. *Theory and Practice of Logic Programming*, *19*(5-6), 1006–1020.

Harrison, A., & Lifschitz, V. (2016). Stable models for infinitary formulas with extensional atoms. *Theory Pract. Log. Program.*, *16*(5-6), 771–786.

Kovács, L., & Voronkov, A. (2013). First-order theorem proving and Vampire. In Sharygina, N., & Veith, H. (Eds.), *Proceedings of the Twenty-fifth International Conference on Computer Aided Verification (CAV'13)*, Vol. 8044 of *Lecture Notes in Computer Science*, pp. 1–35. Springer-Verlag.

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

Lee, J., & Meng, Y. (2012). Stable models of formulas with generalized quantifiers (preliminary report). In Dovier, A., & Santos Costa, V. (Eds.), *Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12)*, Vol. 17, pp. 61–71. Leibniz International Proceedings in Informatics (LIPIcs).

Lee, J., & Meng, Y. (2011). First-order stable model semantics and first-order loop formulas. *J. Artif. Intell. Res.*, *42*, 125–180.

Lifschitz, V. (2019). *Answer Set Programming*. Springer-Verlag.

Lifschitz, V. (2022). Strong equivalence of logic programs with counting. *Theory and Practice of Logic Programming*, *22*(4), 573–588.

Lifschitz, V., Pearce, D., & Valverde, A. (2001). Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, *2*(4), 526–541.

Lifschitz, V., Schaub, T., & Woltran, S. (2018). Interview with vladimir lifschitz. *KI - Künstliche Intelligenz*, *32*(2), 213–218.

Pearce, D., & Valverde, A. (2005). A first order nonmonotonic extension of constructive logic. *Studia Logica*, *30*(2-3), 321–346.

Pelov, N., Denecker, M., & Bruynooghe, M. (2007). Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming*, *7*(3), 301–353.

Peters, S., & Westerstahl, D. (2006). *Quantifiers in language and logic*. OUP Oxford.

Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., & Leone, N. (2012). Team-building with answer set programming in the gioia-tauro seaport. *Theory and Practice of Logic Programming*, *12*(3), 361–381.

Simons, P., Niemelä, I., & Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, *138*(1-2), 181–234.

Son, T., & Pontelli, E. (2007). A constructive semantic characterization of aggregates in answer set programming. *Theory and Practice of Logic Programming*, *7*(3), 355–375.

Truszczyński, M. (2012). Connecting first-order ASP and the logic FO(ID) through reducts. In Erdem, E., Lee, J., Lierler, Y., & Pearce, D. (Eds.), *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, Vol. 7265 of *Lecture Notes in Computer Science*, pp. 543–559. Springer-Verlag.

Vanbesien, L., Bruynooghe, M., & Denecker, M. (2022). Analyzing semantics of aggregate answer set programming using approximation fixpoint theory. *Theory and Practice of Logic Programming*, *22*(4), 523–537.