Recursive Aggregates as Intensional Functions in Answer Set Programming: Semantics and Strong Equivalence

Jorge Fandinno, Zachary Hansen

University of Nebraska Omaha, Omaha, NE, USA
{jfandinno,zachhansen}@unomaha.edu

Abstract

This paper shows that the semantics of programs with aggregates implemented by the solvers `clingo` and `dlv` can be characterized as extended First-Order formulas with intensional functions in the logic of Here-and-There. Furthermore, this characterization can be used to study the strong equivalence of programs with aggregates under either semantics. We also present a transformation that reduces the task of checking strong equivalence to reasoning in classical First-Order logic, which serves as a foundation for automating this procedure.

Introduction

Answer set programming (ASP) is a declarative programming paradigm well-suited for solving knowledge-intensive search and optimization problems (Lifschitz 2019). Its success relies on the combination of a rich knowledge representation language with effective solvers. Some of its most useful constructs are *aggregates*, that is, functions that apply to sets. The semantics of aggregates have been extensively studied in the literature (Simons, Niemelä, and Soininen 2002; Dovier, Pontelli, and Rossi 2003; Pelov, Denecker, and Bruynooghe 2007; Son and Pontelli 2007; Ferraris 2011; Faber, Pfeifer, and Leone 2011; Gelfond and Zhang 2014, 2019; Cabalar et al. 2019). In most cases, they rely on the idea of *grounding*—a process that replaces all variables by variable-free terms. This makes reasoning about First-Order (FO) programs with aggregates cumbersome and it does not allow the use of classical FO theorem provers for verifying properties about this class of programs.

Though several approaches describe the semantics of aggregates bypassing the need for grounding, most of these approaches only allow a restricted class of aggregates (Lee, Lifschitz, and Palla 2008; Lifschitz 2022) or use some extension of the logical language (Bartholomew, Lee, and Meng 2011; Lee and Meng 2012; Asuncion et al. 2015; Cabalar et al. 2018). Recently, Fandinno, Hansen, and Lierler (2022, 2024) showed how to translate logic programs with aggregates into FO sentences, which, after the application of the SM operator (Ferraris, Lee, and Lifschitz 2011), captures the `ASP-Core-2` semantics. Though most practical problems can be represented within the restrictions of the `ASP-Core-2` semantics, some notable exceptions are more naturally represented using recursive aggregates, which are not allowed by `ASP-Core-2`. One of these examples is the *Company Control problem*, which consists of finding companies that control other companies by (directly or indirectly) owning a majority of their shares. This problem has been encoded in the literature using the following logic program (Pelov, Denecker, and Bruynooghe 2007; Faber, Pfeifer, and Leone 2011; Mumick, Pirahesh, and Ramakrishnan 1990; Kemp and Stuckey 1991; Ross and Sagiv 1997):

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

where atom `ownsStk(C1,C2,P)` means that company `C1` directly owns `P`% of the shares of company `C2`; `ctrStk(C1,C2,C3,P)` means that company `C1` controls `P`% of the shares of company `C3` through company `C2` that it controls; and `controls(C1,C3)` means that company `C1` controls company `C3`. Another area where allowing recursive aggregates is important is in the study of *strong equivalence* (Lifschitz, Pearce, and Valverde 2001, 2007). The strong equivalence problem consists of determining whether two programs have the same behavior in any context. Even if the programs we are analyzing do not contain recursion, adding some context may introduce it.

In this paper, we show that the translation introduced by Fandinno, Hansen, and Lierler can also be used for programs with recursive aggregates if we interpret functions in an intensional way (Lin and Wang 2008; Cabalar 2011; Lifschitz 2012; Balduccini 2013; Bartholomew and Lee 2019). We focus on the Abstract Gringo (Gebser et al. 2015) generalization of the semantics by Ferraris (2011), which is used in the answer set solver `clingo`, and the semantics by Faber, Pfeifer, and Leone (2011), which is used by `dlv`. We prove that the translation introduced by Fandinno, Hansen, and Lierler coincides with the Abstract Gringo semantics when function symbols representing sets are interpreted according to the semantics for intensional functions by Bartholomew and Lee. For `dlv`, we introduce a similar translation, which uses a second form of

negation. We show how we can use these translations to express the strong equivalence of the two programs and how to reduce this problem to reasoning in classical FO logic.

Preliminaries

We start by reviewing the syntax of programs with aggregates and presenting an extension of the logic of Quantified Here-and-There (Pearce and Valverde 2008) with intensional functions that is suited for programs with aggregates.

Syntax of programs with aggregates. We follow here the presentation by Fandinno, Hansen, and Lierler (2022). 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 one of the special symbols *inf* and *sup*. A program term (or any other expression) is *ground* if it contains no variables. We assume that a total order on ground terms is chosen such that

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

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

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

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

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

where each t_i ($1 \leq i \leq k$) is a program term and each l_i ($1 \leq i \leq m$) is a basic literal. An *aggregate atom* is of form $\texttt{\#op}\{E\} \prec u$ where op is an operation name, E is an aggregate element, $\prec$ is one of the comparison symbols in (1), and u is a program term, called *guard*. We consider operation names count and sum. For example, the expression

$$\texttt{\#sum\{P,C2 : ctrStk(C1,C2,C3,P)\}} > 50$$

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

$$\mathit{Head} \text{ :- } B_1, \ldots, B_n, \tag{6}$$

where *Head* is an atom or symbol $\bot$, and each B_i is a literal.

We call the symbol :- the *rule operator*. We call the left-hand side of the rule operator the *head*, the right-hand side of the rule operator the *body*. When the head of the rule is an atom we call the rule *normal*, and when it is the symbol $\bot$ we call it a *constraint*. When the body of a normal rule is empty, we call the rule a *fact*. A *program* is a set of rules.

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. If Δ is empty, then $\widehat{\texttt{sum}}(\Delta) = 0$.

Though we illustrate the semantics of aggregates using the operation names count and sum, the semantics can be extended to other operation names by adding the appropriate functions $\widehat{\texttt{op}}$ (Fandinno, Hansen, and Lierler 2024).

Many-sorted logic and extended FO formulas. A many-sorted signature consists of symbols of three kinds—*sorts*, *function constants*, and *predicate constants*. A reflexive and transitive *subsort* relation is defined on the set of sorts. A tuple $s_1, \ldots, s_n$ ($n \geq 0$) of *argument sorts* is assigned to every function constant and to every predicate constant; in addition, a *value sort* is assigned to every function constant. Function constants with $n = 0$ are called *object constants*. For every sort, an infinite sequence of *object variables* of that sort is chosen. *Terms* and *atomic formulas* over a (many-sorted) signature σ are defined as usual with the consideration that the sort of a term must be a subsort of the sort of the function or predicate constant of which it is an argument. *Extended First-Order formulas* over σ are formed from atomic formulas and the 0-place connective $\bot$ (falsity) using the unary connective $\hookleftarrow$, the binary connectives $\wedge$, $\vee$, $\rightarrow$ and the quantifiers $\forall$, $\exists$. We define the usual abbreviations: $\neg F$ stands for $F \rightarrow \bot$ and $F \leftrightarrow G$ stands for $(F \rightarrow G) \wedge (G \rightarrow F)$. We have two negation symbols ($\neg$ and $\hookleftarrow$) and both correspond to classical negation in the context of classical FO logic. The symbol $\neg$ represents standard negation in the logic of Here-and-There and corresponds to default negation in logic programs under the clingo semantics, while symbol $\hookleftarrow$ is a new connective and represents default negation under the dlv semantics. *Interpretations*, *sentences*, *theories*, *satisfaction* and *models* are defined as usual with the additional condition that $I \models \hookleftarrow F$ iff $I \not\models F$. A *standard* FO formula (resp. sentence, theory) is a formula (resp. sentence, theory) without the new operator $\hookleftarrow$.

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

- H and I have the same universe for each sort;
- $p^H \subseteq p^I$ for every predicate constant p in $\mathcal{P}$ and $p^H = p^I$ for every predicate constant p not in $\mathcal{P}$; and
- $f^H = f^I$ for every function constant f not in $\mathcal{F}$.

If I is an interpretation of a signature σ then by σ^I we denote the signature obtained from σ by adding, for every element d of a domain $|I|^s$, its *name* d^* as an object constant of sort s. An *ht-interpretation* of σ is a pair $\langle H, I\rangle$, where H and I are interpretations of σ such that $H \preceq^{\mathcal{PF}} I$. (In terms

of many-sorted Kripke models, I is the there-world, and H is the here-world). The satisfaction relation $\models_{ht}$ between an HT-interpretation $\langle H, I\rangle$ of σ and a sentence F over σ^I is defined recursively as follows:

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

If $\langle H, I\rangle \models_{ht} F$ holds, we say that $\langle H, I\rangle$ *ht-satisfies* F and that $\langle H, I\rangle$ is an *ht-model* of F. If it is clear from the context that the $\models_{ht}$ entailment relation is referred to, we will simply say that $\langle H, I\rangle$ *satisfies* F. We say that $\langle H, I\rangle$ *satisfies* a set of sentences Γ if it satisfies every sentence F in Γ.

We write $H \prec^{\mathcal{PF}} I$ if $H \preceq^{\mathcal{PF}} I$ and $H \neq I$. A model I of a set Γ of sentences is called *stable* if there is no $H \prec^{\mathcal{PF}} I$ such that $\langle H, I\rangle$ satisfies Γ. For finite standard theories, this definition of stable models coincides with the definition of one by Bartholomew and Lee (2019) when sets $\mathcal{P}$ and $\mathcal{F}$ respectively contain the intensional predicate and function constants. For (possibly infinite) standard theories with $\mathcal{F} = \emptyset$, each stable model I corresponds to the equilibrium model $\langle I, I\rangle$ by Pearce and Valverde (2008).

Logic Programs With Aggregates as Extended Many-Sorted First-Order Sentences

We present here translations τ^{cli} and τ^{dlv} that turn a program into extended FO sentences with equality over a signature $\sigma(\mathcal{P}, \mathcal{S})$ of *three sorts*; $\mathcal{P}$ and $\mathcal{S}$ are sets of predicate and *set symbols*, respectively. Superscripts cli and dlv refer to the semantics of `clingo` and `dlv`, respectively.

Target Signature. 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. For brevity's sake, each set symbol $E/\mathbf{X}$ is assigned a short name $|E/\mathbf{X}|$. The target signature is of three sorts. The first sort is called the *general sort* (denoted s_{gen}); all program terms are of this sort. The second sort is called the *tuple sort* (denoted s_{tuple}); it contains entities that are *tuples* of objects of the general sort. The third sort is called the *set sort* (denoted s_{set}); it contains entities that are *sets* of elements of the second sort, that is, sets of tuples of objects of the general sort. Signature $\sigma(\mathcal{P}, \mathcal{S})$ contains:

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

We use infix notation in constructing atoms that utilize predicate symbols of comparisons ($>, \geq, <, \leq, \neq$) and the set membership predicate $\in$. Function constants $s^{cli}_{|E/\mathbf{X}|}$ and $s^{dlv}_{|E/\mathbf{X}|}$ are used to represent sets occurring in aggregates for the `clingo` and `dlv` semantics, respectively. Each of these function constants maps an n-tuple of ground terms $\mathbf{x}$ to the set of tuples represented[1] by $E^{\mathbf{X}}_{\mathbf{x}}$. These claims are formalized below.

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

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

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

When discussing a program Π, we assume a signature $\sigma(\mathcal{P}, \mathcal{S})$ such that $\mathcal{P}$ and $\mathcal{S}$ are the sets that contain all predicate symbols and all set symbols occurring in Π, respectively. Furthermore, when it is clear from the context, we write just σ instead of $\sigma(\mathcal{P}, \mathcal{S})$.

Translations. We now describe translations that convert a program into a set of extended FO sentences. We use $\tau^x_{\mathbf{Z}}$ and τ^x to denote the rules that are common to both translations when x is replaced by either cli or dlv. Given a list $\mathbf{Z}$ of global variables in some rule R, we define $\tau^{cli}_{\mathbf{Z}}$ and $\tau^{dlv}_{\mathbf{Z}}$ for all elements of R as follows.

1. for every atomic formula A occurring outside of an aggregate literal, its translation $\tau^x_{\mathbf{Z}} A$ is A itself; $\tau^x_{\mathbf{Z}} \bot$ is $\bot$;
2. for an aggregate atom A of form $\#\mathtt{count}\{E\} \prec u$ or $\#\mathtt{sum}\{E\} \prec u$, its translation $\tau^x_{\mathbf{Z}}$ is the atom
$$count(s^x_{|E/\mathbf{X}|}(\mathbf{X})) \prec u \text{ or } sum(s^x_{|E/\mathbf{X}|}(\mathbf{X})) \prec u$$

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

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

3. for every (basic or aggregate) literal of the form $not\ A$ its translation $\tau^{cli}_{\mathbf{Z}}(not\ A)$ is $\neg\tau^{cli}_{\mathbf{Z}}A$ and its translation $\tau^{dlv}_{\mathbf{Z}}(not\ A)$ is $\backsim\tau^{dlv}_{\mathbf{Z}}A$; for every literal of the form $not\ not\ A$ its translation $\tau^{cli}_{\mathbf{Z}}(not\ not\ A)$ is $\neg\neg\tau^{cli}_{\mathbf{Z}}A$ and its translation $\tau^{dlv}_{\mathbf{Z}}(not\ not\ A)$ is $\backsim\backsim\tau^{dlv}_{\mathbf{Z}}A$.

We now define the translation τ^x as follows:

4. for every rule R of form (4), its translation $\tau^x R$ is the universal closure of

$$\tau^x_{\mathbf{Z}}B_1 \wedge \cdots \wedge \tau^x_{\mathbf{Z}}B_n \rightarrow \tau^x_{\mathbf{Z}}\mathit{Head},$$

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

5. for every program Π, its translation $\tau^x\Pi$ is the theory containing $\tau^x R$ for each rule R in Π.

τ^{cli} and τ^{dlv} only differ in the translation of negation and the use of different function constants for set symbols. For instance, rule (3) is translated into the universal closure of

$$\begin{aligned} &\mathit{company}(C_1) \wedge \mathit{company}(C_3) \\ &\quad \wedge \mathit{sum}(s^x_{ctr}(C_1, C_3)) > 50 \rightarrow \mathit{controls}(C_1, C_3) \end{aligned} \tag{7}$$

where variables C_1 and C_3 are of the general sort, and x is either cli or dlv depending on the semantics considered.

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

1. universe $|I|^{s_{gen}}$ is the set containing all ground terms of the general sort;
2. universe $|I|^{s_{tuple}}$ is the set of all tuples of form $\langle d_1, \ldots, d_k\rangle$ with $d_i \in |I|^{s_{gen}}$ for each set symbol $E/\mathbf{X}$ in $\mathcal{S}$ with E of the form of (5);
3. every element of $|I|^{s_{set}}$ is a subset of $|I|^{s_{tuple}}$;
4. I interprets each ground program term as itself;
5. I interprets predicate symbols $>, \geq, <, \leq$ according to the total order chosen earlier;
6. I interprets each tuple term of form $\mathit{tuple}(t_1, \ldots, t_k)$ as the tuple $\langle t^I_1, \ldots, t^I_k\rangle$;
7. $\in^I$ is the set of pairs (t, s) s.t. tuple t belongs to set s;
8. for term t_{set} of sort s_{set}, $\mathit{count}(t_{set})^I$ is $\widehat{\texttt{count}}(t^I_{set})$;
9. for term t_{set} of sort s_{set}, $\mathit{sum}(t_{set})^I$ is $\widehat{\texttt{sum}}(t^I_{set})$;

An *agg-interpretation* is a standard interpretation I satisfying, for every set symbol $E/\mathbf{X}$ in $\mathcal{S}$ with E of the form of (5) and for all $x \in \{cli, dlv\}$, that $s^x_{|E/\mathbf{X}|}(\mathbf{x})^I$ is the set of all tuples of the form $\langle (t_1)^{\mathbf{XY}}_{\mathbf{xy}}, \ldots, (t_k)^{\mathbf{XY}}_{\mathbf{xy}}\rangle$ such that I satisfies $\tau^x(l_1)^{\mathbf{XY}}_{\mathbf{xy}} \wedge \cdots \wedge \tau^x(l_m)^{\mathbf{XY}}_{\mathbf{xy}}$.

For instance, the program representing the Company Control problem has a unique set symbol that is associated with the function symbols $s^x_{ctr}/2$ ($x \in \{cli, dlv\}$). If I is an agg-interpretation such that ctrStk^I is the set containing $(c_1, c_2, c_3, 10)$ and $(c_1, c_4, c_3, 20)$, it follows that $s^x_{ctr}(c_1, c_3)^I$ (with $x \in \{cli, dlv\}$) is the set containing tuples $\langle 10, c_2\rangle$ and $\langle 20, c_4\rangle$.

An ht-interpretation $\langle H, I\rangle$ is said to be *standard* if both H and I are standard. An *agg-ht-interpretation* is a standard ht-interpretation $\langle H, I\rangle$ satisfying that I is an agg-interpretation and the following conditions for every set symbol $E/\mathbf{X}$ in $\mathcal{S}$ with E of the form of (5):

- $s^{cli}_{|E/\mathbf{X}|}(\mathbf{x})^H$ is the set of all tuples of form $\langle (t_1)^{\mathbf{XY}}_{\mathbf{xy}}, \ldots, (t_k)^{\mathbf{XY}}_{\mathbf{xy}}\rangle$ such that $\langle H, I\rangle$ satisfies $\tau^{cli}(l_1)^{\mathbf{XY}}_{\mathbf{xy}} \wedge \cdots \wedge \tau^{cli}(l_m)^{\mathbf{XY}}_{\mathbf{xy}}$; and
- $s^{dlv}_{|E/\mathbf{X}|}(\mathbf{x})^H$ is the set of all tuples of form $\langle (t_1)^{\mathbf{XY}}_{\mathbf{xy}}, \ldots, (t_k)^{\mathbf{XY}}_{\mathbf{xy}}\rangle$ such that H satisfies $\tau^{dlv}(l_1)^{\mathbf{XY}}_{\mathbf{xy}} \wedge \cdots \wedge \tau^{dlv}(l_m)^{\mathbf{XY}}_{\mathbf{xy}}$.

where $\mathbf{Y}$ is the lexicographically ordered list of the variables occurring in E that are not in $\mathbf{X}$. Let us consider now an agg-ht-interpretation $\langle H, I\rangle$ where I is as described above and ctrStk^H is the set containing $(c_1, c_2, c_3, 10)$. Then, $s^x_{ctr}(c_1, c_3)^H$ is the set containing tuples $\langle 10, c_2\rangle$. In this example, there is no difference between the semantics of `clingo` and `dlv`. As an example of where these semantics differ, consider an agg-ht-interpretation $\langle H, I\rangle$ with $p^H = r^H = \emptyset$ and $p^I = q^I = q^H = r^I = \{1\}$, and rule

```
p(1) :- #sum{X : q(X), not r(X)} < 1.
```  (8)

This rule is translated into the sentences

$$\mathit{sum}(s^{cli}_{qr}) < 1 \rightarrow p(1) \tag{9}$$

$$\mathit{sum}(s^{dlv}_{qr}) < 1 \rightarrow p(1) \tag{10}$$

for the `clingo` and `dlv` semantics, respectively. It is clear that I satisfies both rules because 1 belongs to p^I. However, when considering the agg-ht-interpretation $\langle H, I\rangle$, only the second rule is satisfied. On the one hand, $(s^x_{qr})^I$ (with $x \in \{cli, dlv\}$) is the empty set. Furthermore, $(s^{cli}_{qr})^H$ is also the empty set because $\langle H, I\rangle \not\models \neg r(1)$, and the antecedent of (9) is satisfied. Then, the rule is not satisfied because the consequent is not satisfied due to 1 not belonging to p^H. On the other hand, $(s^{dlv}_{qr})^H$ is the set containing 1 because $H \models q(1) \wedge \backsim r(1)$. Hence, $\langle H, I\rangle$ does not satisfy the antecedent of (10) and the rule is satisfied.

We now define stable models for programs with aggregates. A model of a formula or theory that is also an agg-interpretation is called an *agg-model* and an agg-ht-interpretation that satisfies a formula or theory is called an *agg-ht-model*.

Definition 1. *An agg-model I of Γ is an $\langle\mathcal{P}, \mathcal{F}\rangle$-*agg-stable model *of Γ if there is no agg-ht-model $\langle H, I\rangle$ with $H \prec^{\mathcal{PF}} I$.*

Correspondence with `clingo` and `dlv`

We establish now the correspondence between the semantics of programs with aggregates introduced in the previous section and the semantics of the solver `clingo`, named Abstract Gringo (Gebser et al. 2015), and the solver `dlv`, which is based on the FLP-reduct (Faber, Pfeifer, and Leone 2011). These semantics are stated in terms of infinitary formulas following the work by Harrison and Lifschitz (2019).

Infinitary Formulas. We extend the definitions of infinitary logic (Truszczyński 2012) to formulas with intensional functions and the $\backsim$ connective. For every nonnegative integer r, *infinitary ground formulas of rank* r are defined recursively:

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

We extend the satisfaction relation for ht-interpretations to infinitary formulas by adding the following two conditions to the definition for FO formulas:

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

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

Truszczyński (2012) defines the satisfaction of infinitary formulas with respect to sets of ground atoms instead of FO interpretations. Such a satisfaction relation for infinitary formulas can be defined when we have no function symbols from $\mathcal{F}$. An infinitary ground formula is *propositional* if it does not contain function symbols from $\mathcal{F}$. For a signature σ, by σ^p we denote the set of all ground atoms over σ that do not contain function symbols from $\mathcal{F}$. Subsets of a propositional signature σ^p are called *propositional interpretations*. The satisfaction relation between a propositional interpretation $\mathcal{A}$ and an infinitary propositional formula is defined recursively:

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

In the following, if I is an interpretation, then $\mathcal{I}$ denotes the set of atomic formulas of σ^p satisfied by I. With this notation, the following result is easily proved by induction.[2]

Proposition 1. *Let F be an infinitary propositional formula. Then, $I \models F$ iff $\mathcal{I} \models F$.*

Grounding. The *grounding* of a FO sentence allows us to replace quantifiers with infinitary conjunctions and disjunctions. Formally, the *grounding of a First-Order sentence F with respect to an interpretation I and sets $\mathcal{P}$ and $\mathcal{F}$* is defined as follows:

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

For a first-order theory Γ, we define $gr_I^{\mathcal{PF}}(\Gamma) = \{gr_I^{\mathcal{PF}}(F) \mid F \in \Gamma\}^\wedge$. For any first-order theory Γ, $gr_I^{\mathcal{PF}}(\Gamma)$ is an infinitary formula, which may contain intensional functions or the $\backsim$ connective. We write $gr_I(\cdot)$ instead of $gr_I^{\mathcal{PF}}(\cdot)$ when it is clear from the context.

Proposition 2. $\langle H, I\rangle \models_{ht} F$ *iff* $\langle H, I\rangle \models_{ht} gr_I(F)$.

Standard formulas and minimal models. We say that an infinitary propositional formula is *standard* if it does not contain the $\backsim$ connective. The definitions of the semantics of `clingo` and `dlv` only use standard infinitary formulas and rely on the notion of minimal models. A propositional interpretation $\mathcal{A}$ satisfies a set Γ of formulas, in symbols $\mathcal{A} \models \Gamma$, if it satisfies every formula in Γ. We say that a set $\mathcal{A}$ of atoms is a $\subseteq$-minimal model of a set of infinitary formulas Γ, if $\mathcal{A} \models \Gamma$ and there is no $\mathcal{B}$ satisfying $\mathcal{B} \models \Gamma$ and $\mathcal{B} \subset \mathcal{A}$.

Clingo. The *FT-reduct* $F^{\mathcal{A}}$ of a standard infinitary formula F with respect to a propositional interpretation $\mathcal{A}$ is defined recursively. If $\mathcal{A} \not\models F$ then $F^{\mathcal{A}}$ is $\bot$; otherwise,

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

We say that a propositional interpretation $\mathcal{A}$ is an *FT-stable model* of a formula F if it is a $\subseteq$-minimal model of $F^{\mathcal{A}}$. We say that a set $\mathcal{A}$ of ground atoms is a *clingo answer set* of a program Π if $\mathcal{A}$ is an FT-stable model of $\tau\Pi$ where τ is the translation from logic programs to infinitary formulas defined by Gebser et al. (2015). The following result states that the usual relation between ht-interpretations and the FT-reduct is satisfied in our settings.

Proposition 3. *Let F be a standard infinitary formula over σ^p. Then, $\langle H, I\rangle \models_{ht} F$ iff $\mathcal{H} \models F^{\mathcal{I}}$.*

For agg-ht-interpretations we can state the relation $\prec^{\mathcal{PF}}$ in terms of the atomic formulas satisfied by it as follows:

Proposition 4. *Let $\langle H, I\rangle$ be an agg-ht-interpretation. Then, $H \prec^{\mathcal{PF}} I$ iff $\mathcal{H} \subset \mathcal{I}$.*

Using Propositions 1-4, we can prove the relation between clingo answer sets and agg-stable models of the corresponding FO theory. Note that clingo answer sets are propositional

[2] Proofs can be found in arixv (Fandinno and Hansen 2024).

interpretations while agg-stable models of FO theories are FO interpretations. To fill this gap, we introduce the following notation. If I is an $\langle \mathcal{P}, \mathcal{S} \rangle$-agg-stable model of $\tau^{cli}\Pi$, we say that $\mathcal{I}$ is a *fo-clingo answer set* of Π.

Theorem 5. *The fo-clingo answer sets of any program coincide with its clingo answer sets.*

Proof sketch. The core of the proof consists of showing that $\langle H, I \rangle \models_{ht} \tau^{cli}\Pi$ iff $\mathcal{H} \models (\tau\Pi)^{\mathcal{I}}$ holds. By Proposition 2, we get $\langle H, I \rangle \models_{ht} \tau^{cli}\Pi$ iff $\langle H, I \rangle \models_{ht} gr_I(\tau^{cli}\Pi)$. Note that $gr_I(\tau^{cli}\Pi)$ is not an infinitary propositional formula, because it may contain function symbols from $\mathcal{F}$. Thus, we cannot apply Proposition 3 directly. However, we can prove that $\langle H, I \rangle \models_{ht} gr_I(\tau^{cli}\Pi)$ iff $\langle H, I \rangle \models_{ht} \tau\Pi$ holds and use Proposition 3 to prove the stated result. Finally, Proposition 4 is used to state the correspondence between stable models of $\tau\Pi$ and agg-stable models of $\tau^{cli}\Pi$. □

The `dlv` semantics. Similarly to the Abstract Gringo semantics, the `dlv` semantics can be stated in terms of the same translation τ to infinitary formulas, but using a different reduct (Harrison and Lifschitz 2019). Let F be an implication $F_1 \to F_2$. Then, the *FLP-reduct* $FLP(F, \mathcal{A})$ of F w.r.t. a propositional interpretation $\mathcal{A}$ is F if $\mathcal{A} \models F_1$, and $\top$ otherwise. For a conjunction of implications $\mathcal{F}^\wedge$, we define

$$FLP(\mathcal{F}^\wedge, \mathcal{A}) \;=\; \{FLP(F, \mathcal{A}) \mid F \in \mathcal{F}\}^\wedge$$

A set $\mathcal{A}$ of ground atoms is an *FLP-stable model* of F if it is a $\subseteq$-minimal model of $FLP(F, \mathcal{A})$. We say that a set $\mathcal{A}$ of ground atoms is a *dlv answer set* of a program Π if $\mathcal{A}$ is an FLP-stable model of $\tau\Pi$. If I is an $\langle \mathcal{P}, \mathcal{S} \rangle$-agg-stable model of $\tau^{dlv}\Pi$, we say that $\mathcal{I}$ is a *fo-dlv answer set* of Π.

Theorem 6. *The fo-dlv answer sets of any program coincide with its dlv answer sets.*

Proof sketch. The structure of the proof is analogous to the one of Theorem 5. Here, the key step of the proof consists of showing $\langle H, I \rangle \models_{ht} \tau^{dlv}\Pi$ iff both $\mathcal{I} \models \tau\Pi$ and $\mathcal{H} \models FLP(\tau\Pi, \mathcal{I})$. □

Strong Equivalence

We say that two programs Π_1 and Π_2 are *strongly equivalent for* `clingo` if program $\Pi_1 \cup \Delta$ and $\Pi_2 \cup \Delta$ have the same `clingo` answer sets for any program Δ.

We assume a signature $\sigma(\mathcal{P}, \mathcal{S})$ where $\mathcal{P}$ and $\mathcal{S}$ are the sets that respectively contain all predicate and all set symbols occurring in $\Pi_1 \cup \Pi_2$.

Theorem 7. *The following conditions are equivalent:*

- Π_1 *and* Π_2 *are strongly equivalent for* `clingo`*;*
- $\tau^{cli}(\Pi_1)$ *and* $\tau^{cli}(\Pi_2)$ *have the same agg-ht-models.*

Let us consider the program formed by rule (8). This program has $\{p(1)\}$ as its unique clingo answer set. Similarly, the program formed by the rule

```
p(1) :- not #sum{X : q(X), not r(X)} >= 1.
```  (11)

also has $\{p(1)\}$ as its unique clingo answer set. However, these two programs are not strongly equivalent under the `clingo` semantics. To illustrate this claim, consider an agg-ht-interpretation $\langle H, I \rangle$ with $p^H = r^H = r^I = \emptyset$, and $p^I = q^H = \{1\}$, and $q^I = \{1, -1\}$. On the one hand, $\langle H, I \rangle$ satisfies (9) because its antecedent is not satisfied as we have $sum(s^{cli}_{qr})^H = 1$. On the other hand, $\langle H, I \rangle$ does satisfy the formula

$$\neg sum(s^{cli}_{qr}) \geq 1 \to p(1) \tag{12}$$

obtained by applying τ^{cli} to (11). Note that in the scope of negation, we only look at the value in I and we have $sum(s^{cli}_{qr})^I = 0$. By Theorem 7, this implies that the two programs are not strongly equivalent under the `clingo` semantics. This assertion can be confirmed by adding context

```
q(1).   q(-X) :- p(X).   :- not p(1).
```  (13)

When added to rule (8), the resulting program has no clingo answer sets, but when added to rule (11), the resulting program has $\{q(1),\ q(-1),\ p(1)\}$ as its unique answer set.

Similarly, we say that two programs Π_1 and Π_2 are *strongly equivalent for* `dlv` if programs $\Pi_1 \cup \Delta$ and $\Pi_2 \cup \Delta$ have the same `dlv` answer sets for any program Δ.

Theorem 8. *The following conditions are equivalent:*

- Π_1 *and* Π_2 *are strongly equivalent for* `dlv`*;*
- $\tau^{dlv}(\Pi_1)$ *and* $\tau^{dlv}(\Pi_2)$ *have the same agg-ht-models.*

Though programs containing rules (8) and (11) are not strongly equivalent for `clingo`, they are strongly equivalent for `dlv`. Applying τ^{dlv} to rule (11) yields formula

$$\backsim sum(s^{dlv}_{qr}) \geq 1 \to p(1) \tag{14}$$

and any agg-ht-interpretation $\langle H, I \rangle$ satisfies the antecedent of (14) iff $H \not\models sum(s^{dlv}_{qr}) \geq 1$ and $I \not\models sum(s^{dlv}_{qr}) \geq 1$ iff $H \models sum(s^{dlv}_{qr}) < 1$ and $I \models sum(s^{dlv}_{qr}) < 1$ iff $\langle H, I \rangle$ satisfies $sum(s^{dlv}_{qr}) < 1$, which is the antecedent of (10). By Theorem 8, this implies that the two programs are strongly equivalent for `dlv`.

We can also use these translations to establish strong equivalence results across the two semantics.

Theorem 9. *The following conditions are equivalent:*

- $\tau^{cli}(\Pi_1)$ *and* $\tau^{dlv}(\Pi_2)$ *have the same agg-ht-models.*
- *the clingo answer sets of* $\Pi_1 \cup \Delta$ *coincide with the dlv answer sets of* $\Pi_2 \cup \Delta$ *for every set* Δ *of rules such that* $\tau^{cli}(\Delta)$ *and* $\tau^{dlv}(\Delta)$ *have the same agg-ht-models.*

In particular, note that $\tau^{cli}(\Delta)$ and $\tau^{dlv}(\Delta)$ are equivalent for any set of rules without aggregates nor double negation. The restriction on the agg-models of $\tau^{cli}(\Delta)$ and $\tau^{dlv}(\Delta)$ is necessary because the same added rules may have different behavior under the two semantics. As an example, consider program Π_1 formed by rule (8) plus constraint

```
:- q(X), r(X).
```  (15)

and program Π_2 formed by rule (11) plus constraint (15). Recall that τ^{cli}(8) is sentence (9) and τ^{dlv}(11) is sentence (14). As we discussed above, any agg-ht-interpretation satisfies the antecedent of (14) iff it satisfies $sum(s^{dlv}_{qr}) <1$.

Hence, it is enough to show $(s^{cli}_{qr})^H = (s^{dlv}_{qr})^H$ for every agg-ht-interpretation $\langle H, I\rangle$ that satisfies $\forall X \neg(q(X) \wedge r(X))$. Every such $\langle H, I\rangle$ satisfies that $H \models q(c)$ implies $H \not\models r(c)$ for every object constant c of general sort. Hence $\langle H, I\rangle$ satisfies $q(c) \wedge \neg r(c)$ iff H satisfies $q(c) \wedge \hookleftarrow r(c)$ for every object constant c of general sort. This implies $(s^{cli}_{qr})^H = (s^{dlv}_{qr})^H$. It is well-known that for programs where all aggregate literals are positive, the `clingo` and `dlv` semantics coincide (Ferraris 2011; Harrison and Lifschitz 2019). As illustrated by this example, Theorem 9 enables us to prove this correspondence for some programs with non-positive aggregates.[3]

Strong Equivalence using Classical Logic

In this section, we show how an additional syntactic transformation γ allows us to replace the logic of Here-and-There by classical FO theory. This also allows us to remove the non-standard negation $\hookleftarrow$ and replace the semantic condition that characterizes agg-interpretations in favor of axiom schemata. This is a generalization of the transformation by Fandinno and Lifschitz (2023) and it is similar to the one used by Bartholomew and Lee (2019) to define the SM operator for FO formulas with intensional functions.

We define a signature $\hat{\sigma}$ that is obtained from the signature σ by adding, for every predicate symbol p other than comparison symbols (4), a new predicate symbol $\hat{p}$ of the same arity and sorts; and for every function symbol $s^x_{|E/\mathbf{X}|}$ with $x \in \{cli, dlv\}$, a new function symbol $\hat{s}^x_{|E/\mathbf{X}|}$.

For any expression E of signature σ, by $\hat{E}$ we denote the expression of $\hat{\sigma}$ obtained from E by replacing every occurrence of every predicate symbol p by $\hat{p}$ and every occurrence of function symbol $s^x_{|E/\mathbf{X}|}$ by $\hat{s}^x_{|E/\mathbf{X}|}$. The translation γ, which relates the logic of here-and-there to classical logic, maps formulas over σ to formulas over $\hat{\sigma}$. It is defined recursively:

- $\gamma F = F \wedge \hat{F}$ if F is atomic,
- $\gamma(\neg F) = \neg\hat{F}$,
- $\gamma(\hookleftarrow F) = \neg\hat{F} \wedge \neg n(F)$,
- $\gamma(F \otimes G) = \gamma F \otimes \gamma G$ with $\otimes \in \{\wedge, \vee\}$.
- $\gamma(F \to G) = (\gamma F \to \gamma G) \wedge (\hat{F} \to \hat{G})$,
- $\gamma(\forall X\, F) = \forall X\, \gamma F$,
- $\gamma(\exists X\, F) = \exists X\, \gamma F$.

where $n(F)$ is the result of replacing all occurrences of $\hookleftarrow$ by $\neg$ in F. To apply γ to a set of formulas means to apply γ to each of its members. Note that $\gamma\Gamma$ is always a standard FO theory (without the $\hookleftarrow$ connective) over the signature $\hat{\sigma}$.

For any ht-interpretation $\langle H, I\rangle$ of σ, I^H stands for the interpretation of $\hat{\sigma}$ that has the same domain as I, interprets symbols not in $\mathcal{P} \cup \mathcal{F}$ in the same way as I, interprets the other function symbols as $f^{I^H} = f^H$ and $\hat{f}^{I^H} = f^I$, and other predicate constants as follows:

$$I^H \models p(\mathbf{d}^*) \text{ iff } H \models p(\mathbf{d}^*); \qquad I^H \models \hat{p}(\mathbf{d}^*) \text{ iff } I \models p(\mathbf{d}^*).$$

[3]Recall that an aggregate literal is called *positive* if it is not in the scope of negation and negation does not occur within its scope (Harrison and Lifschitz 2019).

Proposition 10. $\langle H, I\rangle \models_{ht} \Gamma$ *iff* $I^H \models \gamma\Gamma$.

The following set of formulas characterizes which interpretations of the signature $\hat{\sigma}$ correspond to ht-interpretations. By HT we denote the set of all formulas of the form $\forall \mathbf{X}(p(\mathbf{X}) \to \hat{p}(\mathbf{X}))$ for every predicate symbol $p \in \mathcal{P}$. By AGG we denote the set of all sentences of the form

$$\forall \mathbf{X} T\big(T \in \hat{s}^x_{|E/\mathbf{X}|}(\mathbf{X}) \leftrightarrow \exists \mathbf{Y} \hat{F}^x\big) \tag{16}$$

$$\forall \mathbf{X} T\big(T \in s^{cli}_{|E/\mathbf{X}|}(\mathbf{X}) \leftrightarrow \exists \mathbf{Y} \gamma(F^{cli})\big) \tag{17}$$

$$\forall \mathbf{X} T\big(T \in s^{dlv}_{|E/\mathbf{X}|}(\mathbf{X}) \leftrightarrow \exists \mathbf{Y}\, n(F^{dlv})\big) \tag{18}$$

for every $E/\mathbf{X}$ in $\mathcal{S}$ with E of the form of (5) and where

F^{cli} is $T = \mathit{tuple}(t_1, \ldots, t_k) \wedge \tau^{cli}(l_1) \wedge \cdots \wedge \tau^{cli}(l_m)$

F^{dlv} is $T = \mathit{tuple}(t_1, \ldots, t_k) \wedge \tau^{dlv}(l_1) \wedge \cdots \wedge \tau^{dlv}(l_m)$

Proposition 11. *An interpretation of the signature $\hat{\sigma}$ satisfies HT and AGG iff it can be represented in the form I^H for some agg-ht-interpretation $\langle H, I\rangle$.*

We are ready to state the main result of this section showing that we can use classical FO logic to reason about strong equivalence under the `clingo` and `dlv` semantics.

Theorem 12. *Finite programs Π_1 and Π_2 are strongly equivalent under the* `clingo` *semantics iff all standard interpretations of $\hat{\sigma}$ satisfy the sentence*

$$\bigwedge HT \wedge \bigwedge AGG \to (F_1 \leftrightarrow F_2)$$

where F_i is the conjunction of all sentences in $\gamma\tau^{cli}\Pi_i$. The same holds if we replace `clingo` *and τ^{cli} by* `dlv` *and τ^{dlv}.*

Discussion and Conclusions

In this paper, we provided a characterization of the semantics of logic programs with aggregates which bypasses grounding. We focus on the semantics for recursive aggregates used by ASP solvers `clingo` and `dlv`. Our characterization reflects the intuition that aggregates are functions that apply to sets, usually missing in most formal characterizations of aggregates, which treat them as monolithic constructs. To achieve that, we translate logic programs with aggregates into First-Order sentences with intensional functions, establishing a connection between these two extensions of logic programs. We also show how this characterization can be used to study the strong equivalence of programs with aggregates and variables under either semantics. Finally, we show how to reduce the task of checking strong equivalence to reasoning in classical First-Order logic, which serves as a foundation for automating this procedure. We also axiomatize the meaning of the symbols used to represent sets. The axiomatization of the symbols representing aggregate operations (`sum` and `count`) developed by Fandinno, Hansen, and Lierler (2022) for non-recursive aggregates also applies to recursive aggregates because these function symbols stay non-intensional. Immediate future work includes the integration of this characterization of aggregates with the formalization of arithmetics used by the verification tool ANTHEM (Fandinno et al. 2020, 2023) and the implementation of a new verification tool that can accommodate programs with aggregates.

Acknowledgements

This research is partially supported by NSF CAREER award 2338635. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Asuncion, V.; Chen, Y.; Zhang, Y.; and Zhou, Y. 2015. Ordered completion for logic programs with aggregates. *Artificial Intelligence*, 224: 72–102.

Balduccini, M. 2013. ASP with non-herbrand partial functions: A language and system for practical use. *Theory and Practice of Logic Programming*, 13(4-5): 547–561.

Bartholomew, M.; and Lee, J. 2019. First-order stable model semantics with intensional functions. *Artificial Intelligence*, 273: 56–93.

Bartholomew, M.; Lee, J.; and 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)*, 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.; and 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.; and Schellhorn, S. 2019. Gelfond-Zhang aggregates as propositional formulas. *Artificial Intelligence*, 274: 26–43.

Dovier, A.; Pontelli, E.; and Rossi, G. 2003. Intensional Sets in CLP. In Palamidessi, C., ed., *Proceedings of the Nineteenth International Conference on Logic Programming (ICLP'03)*, volume 2916 of *Lecture Notes in Computer Science*, 284–299. Springer-Verlag.

Faber, W.; Pfeifer, G.; and Leone, N. 2011. Semantics and Complexity of Recursive Aggregates in Answer Set Programming. *Artificial Intelligence*, 175(1): 278–298.

Fandinno, J.; and Hansen, Z. 2024. Recursive Aggregates as Intensional Functions in Answer Set Programming: Semantics and Strong Equivalence. arXiv:2412.10975.

Fandinno, J.; Hansen, Z.; and Lierler, Y. 2022. Axiomatization of Aggregates in Answer Set Programming. In *Proceedings of the Thirty-sixth National Conference on Artificial Intelligence (AAAI'22)*, 5634–5641. AAAI Press.

Fandinno, J.; Hansen, Z.; and Lierler, Y. 2024. Axiomatization of Non-Recursive Aggregates in First-Order Answer Set Programming. *Journal of Artificial Intelligence Research*, 80: 977–1031.

Fandinno, J.; Hansen, Z.; Lierler, Y.; Lifschitz, V.; and Temple, N. 2023. External Behavior of a Logic Program and Verification of Refactoring. *Theory and Practice of Logic Programming*, 23(4): 933–947.

Fandinno, J.; and Lifschitz, V. 2023. On Heuer's Procedure for Verifying Strong Equivalence. In Gaggl, S.; Martinez, M.; and Ortiz, M., eds., *Proceedings of the Eighteenth European Conference on Logics in Artificial Intelligence (JELIA'23)*, volume 14281 of *Lecture Notes in Computer Science*. Springer-Verlag.

Fandinno, J.; Lifschitz, V.; Lühne, P.; and Schaub, T. 2020. Verifying Tight Logic Programs with anthem and vampire. *Theory and Practice of Logic Programming*, 20(5): 735–750.

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.; and Lifschitz, V. 2011. Stable models and circumscription. *Artificial Intelligence*, 175(1): 236–263.

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

Gelfond, M.; and Zhang, Y. 2014. Vicious Circle Principle and Logic Programs with Aggregates. *Theory and Practice of Logic Programming*, 14(4-5): 587–601.

Gelfond, M.; and Zhang, Y. 2019. Vicious Circle Principle, Aggregates, and Formation of Sets in ASP Based Languages. *Artificial Intelligence*, 275: 28–77.

Harrison, A.; and Lifschitz, V. 2019. Relating Two Dialects of Answer Set Programming. *Theory and Practice of Logic Programming*, 19(5-6): 1006–1020.

Kemp, D.; and Stuckey, P. 1991. Semantics of Logic Programs with Aggregates. In Saraswat, V.; and Ueda, K., eds., *Proceedings of the 1991 International Symposium on Logic Programming (ISLP'91)*, 387–401. MIT Press.

Lee, J.; Lifschitz, V.; and Palla, R. 2008. A Reductive Semantics for Counting and Choice in Answer Set Programming. In Fox, D.; and Gomes, C., eds., *Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08)*, 472–479. AAAI Press.

Lee, J.; and Meng, Y. 2012. Stable Models of Formulas with Generalized Quantifiers (Preliminary Report). In Dovier, A.; and Santos Costa, V., eds., *Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12)*, volume 17, 61–71. Leibniz International Proceedings in Informatics (LIPIcs).

Lifschitz, V. 2012. Logic Programs with Intensional Functions. In Brewka, G.; Eiter, T.; and McIlraith, S., eds., *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*. AAAI Press.

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.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4): 526–541.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2007. A Characterization of Strong Equivalence for Logic Programs with Variables. In Baral, C.; Brewka, G.; and Schlipf, J., eds., *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*, 188–200. Springer-Verlag.

Lin, F.; and Wang, Y. 2008. Answer Set Programming with Functions. In Brewka, G.; and Lang, J., eds., *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, 454–465. AAAI Press.

Mumick, I.; Pirahesh, H.; and Ramakrishnan, R. 1990. The Magic of Duplicates and Aggregates. In McLeod, D.; Sacks-Davis, R.; and Schek, H., eds., *Proceedings of the Sixteenth International Conference on Very Large Data Bases (VLDB'90)*, 264–277. Morgan Kaufmann Publishers.

Pearce, D.; and Valverde, A. 2008. Quantified Equilibrium Logic and Foundations for Answer Set Programs. In Garcia de la Banda, M.; and Pontelli, E., eds., *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*, 546–560. Springer-Verlag.

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

Ross, K.; and Sagiv, Y. 1997. Monotonic Aggregation in Deductive Databases. *Journal of Computer and System Sciences*, 54(1): 79–97.

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

Son, T.; and 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.; and Pearce, D., eds., *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, 543–559. Springer-Verlag.