# Vicious Circle Principle, Aggregates, and Formation of Sets in ASP Based Languages

Michael Gelfond and Yuanlin Zhang

*Texas Tech University, Lubbock, Texas 79414, USA*
{*michael.gelfond, y.zhang*}@*ttu.edu*

**Abstract**

The paper introduces an extension of the original Answer Set Prolog (ASP) by several set constructs including aggregates, defined as functions on sets. The new language, called $\mathscr{A}log$ allows creating sets based on the Vicious Circle Principle by Poincaré and Russell which eliminates a number of problems found in existing extensions of ASP by aggregates. We argue that, despite the fact that $\mathscr{A}log$ is not as expressive as other extensions of ASP by aggregates, clarity of its syntax and semantics, addition of several new set-based constructs, and simplicity and the ease of use make it a viable competitor to these languages. We also study a number of important properties of the language and show how ideas used in its design can be utilized to generalize and simplify the definition of another important extension of ASP by aggregates.

*Keywords:* Aggregates; Answer Set Programming; Logic Programming; Knowledge Representation; Language Design
*2010 MSC:* 68N17, 68T27, 68T30

## 1. Introduction

The development of answer set semantics for logic programs [1, 2] led to the creation of a powerful knowledge representation language, Answer Set Prolog (ASP) [3], capable of representing recursive definitions, defaults, effects of actions and other important phenomena of natural language. A program of ASP consists of rules understood as constraints on so called *answer sets* – possible collections of beliefs of a rational agent associated with the program. The rule *head* $\leftarrow$ *body* requires the agent who believes the body of the rule to also believe the rule's *head*. In forming its beliefs the agent is supposed to satisfy the rules, avoid contradictions, and adhere to Rationality Principle: *believe nothing you are not forced to believe (by the rules of the program)*. This intuition is captured by the original definition of answer sets [3]. On the theoretical side, this work helped some people to better understand formation of rational beliefs and other forms of non-monotonic reasoning. In addition, the design of algorithms for computing answer sets and their efficient implementations in systems called *ASP solvers* for instance, [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]) allowed the language to become a powerful tool for building non-trivial knowledge intensive applications such as [15, 16, 17, 18] among others. There are a number of extensions of ASP which also contributed to its success. Some, such as CR-prolog [19] and P-log [20], which enhanced ASP by abductive and probabilistic reasoning respectively, preserved the epistemic character of the language. Others, such as [21] and [22] abandoned the original idea in favor of expanding the syntax of the language to include arbitrary formulae and establishing closer relationship with traditional super-intuitionistic and classical logics. Extensions of the original language by other new constructs, such as choice rules, weight constraints, and minimization statement [23, 4], weak constraints [24], etc. were motivated by the desire to use ASP to solve optimization problems and by other practical needs of ASP. In this paper we are especially interested in the large collection of works aimed at expanding ASP with *aggregates* - functions defined on sets of objects of the domain. Here is a typical example of the use of aggregates for knowledge representation.

**Example 1 (Classes That Need Teaching Assistants).** Suppose that we have a complete list of students enrolled in a class *c* that is represented by the following collection of atoms:

```
enrolled(c,mike).
enrolled(c,john).
...
```

Suppose also that we would like to define a new relation $need\_ta(C)$ that holds iff the class $C$ needs a teaching assistant. In this particular school $need\_ta(C)$ is true iff the number of students enrolled in the class is greater than 20. The definition can be given by the following rules in the language of logic programs with aggregates:

```
need_ta(C) :- card{X : enrolled(C,X)} > 20.
-need_ta(C) :- not need_ta(C).
```

where *card* stands for the cardinality function. Let us call the resulting program $TA$. □

The program is simple, has a clear intuitive meaning, and can be run on most of the existing ASP solvers. However, the situation is more complex than that. Unfortunately, currently there is no single recognized language of logic programs with aggregates. Instead there is a comparatively large collection of such languages with different syntax and, even more importantly, different semantics (see [25, 4, 26, 27, 28, 29] among others).

To illustrate the difficulty, consider the programs in the following example.

**Example 2.** Program $P_0$ consists of a rule:

```
p(1) :- card{X: p(X)} != 1.
```

Program $P_1$ consists of rules:

```
p(1) :- p(0).
p(0) :- p(1).
p(1) :- card{X: p(X)} != 1.
```

Program $P_2$ consists of rules:

```
p(1) :- card{X: p(X)} >= 0.
```

Even for these seemingly simple programs, there are different opinions about their meaning. To the best of our knowledge, all ASP based semantics, including that of [27, 26, 30], view $P_0$ as a bad specification. It is inconsistent, i.e., has no answer sets. Opinions differ, however, about the meaning of the other two programs. [27] views $P_1$ as a reasonable specification having one answer set $\{p(0), p(1)\}$. According to [26, 30], $P_1$ is inconsistent.[1] According to most semantics, $P_2$ has one answer set $\{p(1)\}$. However, the same semantics view seemingly equivalent program $P_3$

```
p(1) :- card{X: p(X)} = Y, Y >= 0.
```

as inconsistent. □

In our judgment this and other similar "clashes of intuition" cause a serious impediment to the use of aggregates in ASP (as well as in some other KR languages). It

[1] In the rest of the paper we often refer to languages from [27] and [26] as $\mathscr{F}log$ and $\mathscr{S}log$ respectively.

is, of course, not entirely clear how this type of differences can be resolved. Sometimes, further analysis can find convincing arguments in favor of one of the proposals. Sometimes, the analysis discovers that different approaches really model different linguistic or natural phenomena and are, hence, all useful in different contexts. But, in general, we believe that the difficulty can be greatly alleviated if we pay more serious attention to important principles of language design, such as

- *Naturalesness*: Constructs of a formal language $\mathscr{L}$ should be close to informal constructs used in the parts of natural language that $\mathscr{L}$ is designed to formalize. The language should come with a methodology of using these constructs for knowledge representation and programming.
- *Clarity*: The language should have simple syntax and clear intuitive semantics based on understandable informal principles.
- *Mathematical Elegance*: Formal description of syntax and semantics of the language should be mathematically elegant. Moreover, the language should come with mathematical theory facilitating its use for knowledge representation and programming.
- *Stability*: Informally equivalent transformations of a text should correspond to formally equivalent ones.
- *Elaboration Tolerance*: It should be possible to expand a language by new relevant constructs without substantial changes in its syntax and semantics.

We learned these principles from the early work on language design, especially that by Dijkstra, Hoare, Wirth, and McCarthy, and had multiple opportunities to confirm their importance in our own work.

In this paper we use these principles for the design of a new KR language $\mathscr{A}log$[2] which allows aggregates, an analog of choice rules, and other set related constructs which, in our judgment, make ASP a better tool for knowledge representation. The emphasis on sets is the result of our analysis of previous work on ASP aggregates. We believe, that the above mentioned *"clash of intuitions" is caused not so much by the ambiguity of intuitive meaning of aggregates as by the lack of clear understanding of the meaning and proper ways of formation of the ASP sets*.

[2]The first version of $\mathscr{A}log$ appeared in [30]. The language was further elaborated in [31].

(More discussion of this, and of the relationship between formation of sets in ASP and in classical set theory can be found in Section 2. )

The paper is organized as follows. We start with expanding the original ASP by aggregates and proceed by gradually introducing new set related constructs and giving examples of their use for knowledge representation.[3] The choice for such a step-wise introduction of $\mathscr{A}log$ is aimed to illustrate the concept of elaboration tolerance of a language. In addition, we believe that this allows for separation of concerns which makes it easier to grasp the paper's ideas. In the process, we illustrate the influence of general principles mentioned above on the design of the language and contrast it with decisions made in some other similar languages. Section 2 contains analysis of the process of creation of sets in logic programming languages and its relation with the Vicious Circle Principle (VCP) by Poincaré and Russell. Section 3 introduces an extension of ASP by aggregates understood as functions on (possibly infinite) sets, gives its syntax and semantics, and compares the new language with existing approaches, including that of $\mathscr{F}log$ and $\mathscr{S}log$. In section 4, we give informal semantics of rules with subset relation and discuss its usefulness in knowledge representation. Section 5 contains syntax and semantics of the full language. Section 6 presents a number of important properties of $\mathscr{A}log$ programs whose proofs are contained in the Appendix. In section 7, we use $\mathscr{S}log$ as an example to demonstrate how our newly introduced additive reduct can be used to define and extend other semantics of aggregates in a comparatively simple manner while explicating essential ideas underlying the semantics. Specifically, we extend $\mathscr{S}log$ to allow disjunctions in the head of rule and show that the semantics of the new language agrees with $\mathscr{S}log$ on the programs without disjunction. Section 8 discusses the related work while Section 9 concludes the paper.

As was mentioned earlier basic constructs of $\mathscr{A}log$ presented in this paper were first introduced in two conference publications [30] and [31]. The current paper extends this work by

- adding a number of discussions explaining our general phylosophy of language design and its influence on the development $\mathscr{A}log$,
- providing new examples illustrating the use of the language for knowledge representation as well as substantial differences between $\mathscr{A}log$ and other ASP based languages with aggregates,

[3]The alternative would be to start with the full language and define its semantics using a single notion of a reduct.

- generalizing properties of ASP with aggregates stated in the original papers to the full version of $\mathscr{A}log$,
- substantially improving the version of the Splitting Set Theorem presented in [31] and adding new results related to the notions of aggregate stratification and stability of programs with respect to arithmetic transformations,
- Adding proofs of the propositions stated in the paper.

## 2. Sets of $\mathscr{A}log$

$\mathscr{A}log$ expands Answer Set Prolog by a *set expression*

$$\{\bar{X} : p(\bar{X})\}$$

which denotes *the set of all objects of the program's domain believed by the rational agent associated with the program to satisfy property p*. Note, that to denote sets we use standard mathematical syntax. However, in mathematics $\{\bar{X} : p(\bar{X})\}$ is usually read as "the set of all $\bar{X}$ for which $p(\bar{X})$ is *true*". Our reading is in line with epistemic interpretation of *ASP*. From our standpoint it is difficult to talk about truth of an arbitrary statement $p(t)$ in any non-monotonic logic in which addition of new information can change the truth-value of $p(t)$. We share the classical understanding of truth according to which truth is everlasting and does not change with time or growth of our knowledge.

As we know from the early work on set theory one should be very careful with the use of set expressions, since they may actually denote no sets. For instance, as shown by Russell and Zermelo, no set is denoted by an expression $\{X : X \notin X\}$.

Perhaps somewhat surprisingly, similar phenomena may be observed in logic programming. It is easy to see that no set is denoted by set expression $\{X : p(X)\}$ used in a program $P$ consisting of the rule:

```
p(0) :- card{X:p(X)} = 0.
```

Assuming that 0 is the only object constant of the program, there are two possible answer sets of $P$: $\{\ \}$ and $\{p(0)\}$. The former does not satisfy the rule of $P$. The latter is not sanctioned by any rule and hence does not adhere to the rationality principle. With the absence of answer sets, $\{X : p(X)\}$ has no denotation.

In both, set-theoretic and logic programing examples the difficulty with the existence of sets seem to be connected with self-reference in their definitions.

The problem was recognized in early stages of the development of classical set theory by G. Cantor according to whom

"*A set is a Multiplicity (Many) that allows itself to be thought of as a Unity (One).*"

Apparently, neither of the "multiplicities" from the above examples allow themselves to be thought of as a Unity. There are multiple and still ongoing efforts to better understand when a "Multiplicity" gives itself such a permission. The approach proposed in this paper is greatly influenced by the work of Poincaré and Russell who suggested to prohibit definitions containing vicious circles. This prohibition, which in one of its many formulations says: *"no object or property may be introduced by a definition that depends on that object or property itself,"* is often referred to as the "*Vicious Circle Principle*" (VCP). The semantics of sets in $\mathscr{A}log$ is based on the following adaptation of this principle to Logic Programming.

VCP in $\mathscr{A}log$: *The reasoner's belief in $p(t)$ can not depend on existence of a set denoted by set expression $\{X : p(X)\}$,*

or, equivalently,

*Set expression $\{X : p(X)\}$ denotes a set S only if for every t rational belief in $p(t)$ can be established without a reference to S.*

Clearly, an expression $f\{X : p(X)\}$ where $f$ is an aggregate can only be meaningful with respect to a program $P$ if the program is consistent (and hence $\{X : p(X)\}$ has a denotation).

To further illustrate the intuition behind $\mathscr{A}log$'s version of VCP consider the following example.

**Example 3.** Let us consider programs $P_0 \dots P_3$ from Example 2. $P_0$, consisting of

```
p(1) :- card{X: p(X)} != 1
```

clearly has no answer set since $\{\}$ does not satisfy its rule and there is no justification for believing in $p(1)$. $P_1$, consisting of

```
p(1) :- p(0).
p(0) :- p(1).
p(1) :- card{X: p(X)} != 1.
```

is also inconsistent. To see that notice that the first two rules of the program limit our possibilities to $A_1 = \{\}$ and $A_2 = \{p(0), p(1)\}$. In the first case $\{X : p(X)\}$ denotes $\{\}$. But this contradicts the last rule of the program. $A_1$ cannot be an answer set of $P_1$. In $A_2$, $\{X : p(X)\}$ denotes $S = \{0, 1\}$. But this violates our form of VCP since the reasoner's beliefs in both, $p(0)$ and $p(1)$, cannot be established without reference to $S$. $A_2$ is not an answer set either. Now consider program $P_2$ with rule

```
p(1) :- card{X: p(X)} >= 0.
```

There are two possible answer sets: $A_1 = \{\}$ and $A_2 = \{p(1)\}$. In $A_1$, $S = \{\}$ which contradicts the rule. In $A_2$, $S = \{1\}$ but this would contradict the $\mathscr{A}log$'s VCP. The program is inconsistent.[4] Similar argument shows inconsistency of $P_3$.

Finally, consider program consisting of a rule

```
p(1) :- count{X : p(X), X != 1} = 0
```

Even though this rule contains recursion through aggregates it does not violate VCP since the definition of $p(1)$ does not refer to the denotation of $\{X : p(X)\}$. $\square$

We hope that the examples are sufficient to show how the informal semantics of $\mathscr{A}log$ can give a programmer some guidelines in avoiding formation of sets problematic from the standpoint of VCP.

## 3. Aggregates of $\mathscr{A}log$

For simplicity of presentation we limit our attention to aggregates defined as functions from sets of terms into the set of natural numbers. Similar approach will work for integers, rational numbers, Turing-computable real numbers, etc.

### *3.1. Syntax of Aggregates*

Let $\Sigma$ be a (possibly sorted) signature with a finite collection of predicate and function symbols and (possibly infinite) collection of object constants, and let $\mathscr{A}$ be a finite collection of aggregate names. Terms and literals over signature $\Sigma$ are defined as usual and referred to as *regular*.[5] A literal formed by a predicate

[4] There is a common argument for the semantics in which $\{p(1)\}$ would be the answer set of $P_2$: "Since $card\{X : p(X)\} \geq 0$ is always true it can be dropped from the rule without changing the rule's meaning". But the argument assumes the existence of the set denoted by $\{X : p(X)\}$ which is not always the case in $\mathscr{A}log$.

[5] Recall, that a *negative literal* of ASP is of the form $\neg p$ (read as "$p$ is believed to be false"). It is different from an expression *not* $l$, where *not* is a default negation and $l$ is a regular literal, which is read as "it is not believed that $l$ is true". A literal possibly preceded by default negation is called an *extended literal*.

symbol different from arithmetic predicate and equality is called *user-defined*. To incorporate aggregates into the language of ASP we expand the ASP syntax by

- *Set expressions* – constructs of the form

$$\{\bar{X} : cond\} \tag{1}$$

  where *cond* is a finite collection of regular literals and $\bar{X}$ is a list of variables occurring in *cond*.

- *Set atoms* – statements of the form

$$f(S) \odot k \tag{2}$$

  where $f$ is an aggregate name, $S$ is a set expression, $k$ is a natural number, and $\odot$ is an arithmetic relation $>, \geq, <, \leq, =$ or !=. Typical aggregate names in existing solvers include `count, sum, min`, and `max` which have the intuitive meaning as suggested by their names. When applying to a set of tuples, the value of `sum` is the sum of the first component of all tuples of the set. This agreement allows to nicely avoid dealing with multisets and we follow it in $\mathscr{A}log$. Note that an aggregate $f$ does not have to be total. Hence, $f(S) > k$ holds if $f(S)$ is defined and its value, $y$, is greater than $k$. Similarly for other arithmetic relations. If $f(S)$ is undefined then so is the truth value of the set atom (2).

- *Aggregate rules* – statements of the form

$$head \leftarrow body \tag{3}$$

  where *head* is a disjunction of regular literals and *body* is a collection of regular literals (possibly preceded by *not*) and set atoms. We say that literal *l belongs to* a rule if it is one of the disjuncts in its head or if *l* or *not l* is an element of its body. For instance, literals $l_1$, $l_2$, and $l_3$ belong to rule

$$l_1 \leftarrow l_2, count\{X : p(X)\}, not\ l_3$$

  while literals of the form $p(t)$ where $t$ is an arbitrary term do not.

  Both the head and the body can be *infinite*. As usual, if the head is empty the rule is referred to as a *constraint*. If the head is not empty but the body is we omit $\leftarrow$ and refer to the rule as a *fact*. When describing a program in this paper, we use `:-` for $\leftarrow$.

Infinite rules are introduced together with aggregates because they facilitate description of the semantics of aggregates defined on infinite sets. Regular and set atoms are referred to as *atoms*. An *aggregate program* is a collection of aggregate rules over some signature $\Sigma$. As in the original ASP a rule with variables is viewed as an abbreviation for the collection of its ground instances. Recall that in ASP a rule is called *ground* if it contains no variables and no occurrences of symbols for arithmetic functions. (Similarly for terms, atoms, programs, etc.) A ground rule obtained from an ASP rule $r$ by replacing $r$'s variables with (properly typed) ground terms of the language and by evaluating the rule's arithmetic functions is called a *ground instance* of $r$. The same definitions apply to regular rules of $\mathscr{A}log$. For rules containing set atoms, however, the situation is slightly more complex and requires some additional definitions.

**Definition 1 (Set Variables and Their Bound Occurrences).** Variables from $\bar{X}$ in a set expression (1) are referred to as *set variables*. An occurrence of a set variable in a set expression is called *bound*. □

For instance, the occurrences of $X$ in $\{X : p(X,Y)\}$ are bound while the occurrence of $Y$ is not.

**Definition 2 (Ground Instances).** A rule $r$ is called *ground* if every occurrence of a variable in $r$ is bound and $r$ contains no occurrences of symbols for arithmetic functions. A ground rule obtained from a non-ground rule $r$ by replacing non-bound (*free*) occurrences of $r$'s variables with (properly typed) regular ground terms of the language and by evaluating the rule's arithmetic functions is called a *ground instance* of $r$. □

This definition is illustrated by the following two examples.

**Example 4 (Grounding).** Consider a program $P_4$ with variables

```
q(Y) :- card{X:p(X,Y)} = 1, r(Y).
r(a).  r(b).  p(a,b).
```

(Unless otherwise stated we assume that program signature contains no other constants except those appearing in the program.) All occurrences of the set variable $X$ in $P_4$ are bound; all occurrences of the variable $Y$ are free. The program's grounding, $ground(P_4)$, is

```
q(a) :- card{X:p(X,a)} = 1, r(a).
q(b) :- card{X:p(X,b)} = 1, r(b).
r(a).  r(b).  p(a,b).
```

□

The next example deal with the case when some occurrences of a set variable in a rule are free and some are bound.

**Example 5 (Grounding).** Consider program $P_5$

```
r :- card{X:p(X)} >= 2, q(X).
p(a).  p(b).  q(a).
```

Here the occurrence of $X$ in $p(X)$ is bound but its occurrence in $q(X)$ is free. Hence the ground program $ground(P_5)$ is:

```
r :- card{X:p(X)} >= 2, q(a).
r :- card{X:p(X)} >= 2, q(b).
p(a).  p(b).  q(a).
```

□

**Discussion**: Note that despite its apparent simplicity the syntax of $\mathscr{A}log$ differs substantially from the syntax of many other logic programming languages allowing aggregates. We illustrate the differences using the language $\mathscr{F}log$ [27] which serves as the basis for the treatment of aggregates in a popular ASP reasoning system CLINGO [12]. While syntactically programs of $\mathscr{A}log$ can also be viewed as programs of $\mathscr{F}log$, the opposite is not true. Among other things, $\mathscr{F}log$ allows parameters of aggregates to be substantially more complex than those of $\mathscr{A}log$. For instance, an expression $f\{a : p(a,a), b : p(b,a)\} = 1$, where $f$ is an aggregate, is an atom of $\mathscr{F}log$ but not of $\mathscr{A}log$. This construct, which is different from the usual set-theoretic notation as used in $\mathscr{A}log$, can not be simply ignored since it is important for the $\mathscr{F}log$ definition of grounding. For instance, the grounding of the first rule of $P_4$ from Example 4

```
q(Y) :- card{X:p(X,Y)} = 1, r(Y)
```

understood as a program of $\mathscr{F}log$ consists of $\mathscr{F}log$ rules

```
q(a) :- card{a:p(a,a),b:p(b,a)} = 1, r(a).
q(b) :- card{a:p(a,b),b:p(b,b)} = 1, r(b).
```

which is not even a program of $\mathscr{A}log$. Another important difference between the grounding methods of these languages can be illustrated by program $P_5$ from Example 5:

```
r :- card{X:p(X)} >= 2, q(X).
p(a).  p(b).  q(a).
```

The $\mathscr{F}log$ grounding of $P_5$, $ground_f(P_5)$, is:

```
r :- card{a:p(a)} >= 2, q(a).
r :- card{b:p(b)} >= 2, q(b).
p(a).  p(b).  q(a).
```

Clearly this is substantially different from the $\mathscr{A}log$ grounding of $P_5$:

```
r :- card{X:p(X)} >= 2, q(a).
r :- card{X:p(X)} >= 2, q(b).
p(a).  p(b).  q(a).
```

The difference in grounding reflects important semantic differences between the two languages. It is easy to see that $\mathscr{A}log$'s answer set of $P_5$ is $A_1 = \{p(a), p(b), q(a), r\}$ while according to $\mathscr{F}log$ the same program has different answer set, $A_2 = \{p(a), p(b), q(a)\}$. Because of its definition of grounding, $\mathscr{F}log$ *does not satisfy the stability principle of language design*. One can easily check that *replacement of* $\{X : p(X)\}$ *in* $P_5$ *by an equivalent set expression* $\{Y : p(Y)\}$ *changes the meaning of* $P_5$ *with respect to* $\mathscr{F}log$*'s semantics*. The new program will have the same answer set $A_1$ in both $\mathscr{A}log$ and $\mathscr{F}log$. From the semantics of $\mathscr{A}log$, it will immediately follow that this condition holds for an arbitrary program of the language. In other words, $\mathscr{A}log$ is stable with respect to renaming bound variables.

### *3.2. Semantics of Aggregates*

Since non-ground programs of $\mathscr{A}log$ can be viewed as abbreviations for the collections of their ground instances, it is sufficient to define the semantics for ground programs. As usual, the semantics is given by a program's answer sets – collections of possible beliefs of a rational reasoner associated with the program.

Let us first notice that the *original definition of answer sets from [3] is applicable to programs with infinite rules*. Hence we already have the definition of answer sets for aggregate programs not containing occurrences of set atoms.

Now let us extend the definition to aggregate programs. First recall that literals $p(t)$ and $\neg p(t)$ are called *contrary* and that $\bar{l}$ denotes the literal contrary to $l$.

**Definition 3 (Satisfiability of Aggregate Rules).** Let $A$ be a set of ground regular literals.

- if $l$ is a regular literal then
  - $l$ is *true* in $A$ if $l \in A$.
  - $l$ is *false* in $A$ if $\bar{l} \in A$.
  - $l$ is *undefined* in $A$ if neither $l$ nor $\bar{l}$ is in $A$.
  - *not* $l$ is *true* in $A$ if $l \notin A$. Otherwise, *not* $l$ is *false* in $A$.
  - a disjunction of regular literals is *true* in $A$ if at least one of its members is true in $A$.
- If $l$ is of the form $f\{\bar{X} : cond\} \odot k$ then we have two cases:
  - $f\{\bar{t} : cond(\bar{t}) \subseteq A\}$ is defined and has the value $y$. Then
    - if an arithmetic statement $y \odot k$ is true then $f\{\bar{X} : cond\} \odot k$ is *true* in $A$.
    - if $y \odot k$ is false then $f\{\bar{X} : cond\} \odot k$ is *false* in $A$.
  - $f\{\bar{t} : cond(\bar{t}) \subseteq A\}$ is undefined. Then $f\{\bar{X} : cond\} \odot k$ is *undefined* in $A$.
- A rule is *satisfied* by $A$ if its head is *true* in $A$ or at least one of a set atoms or extended literals in its body is *false* or *undefined* in $A$.

In what follows we treat "true in $A$" and "satisfied by $A$" as synonyms. □

For instance, atom $card\{X : p(X)\} \geq 0$ is undefined in $A$ if $A$ contains an infinite collection of atoms formed by $p$.

The main technical tool used to define semantics of aggregates is that of *aggregate reduct*. Unlike other existing ASP reducts which normally remove a program's rule or some extended literal from the rule's body, the aggregate reduct may replace a rule by a new one in which an aggregate atom $f\{X : cond\} \odot k$ is replaced by a possibly infinite collection of regular atoms representing the set denoted by $\{X : cond\}$. We will call reducts which may add new atoms to the rules of the program *additive*. (More information on additive reducts will be given in section 7.)

**Definition 4 (Reduct for Aggregate Programs).** Let $\Pi$ be a ground aggregate program. The *aggregate reduct* of $\Pi$ with respect to a set of ground regular literals $A$ is obtained from $\Pi$ by

1. removing rules containing set atoms which are *false* or *undefined* in $A$.
2. replacing every remaining set atom $f\{\bar{X} : cond\} \odot k$ by

$$\cup_{cond(\bar{t}) \subseteq A} cond(\bar{t}).$$

□

The first clause of the definition removes rules that are useless because of the truth values of their aggregates in $A$. The next clause reflects the principle of avoiding vicious circles. Clearly, aggregate reducts do not contain set atoms.

**Definition 5 (Answer Sets).** A set $A$ of ground regular literals over the signature of a ground aggregate program $\Pi$ is an *answer set* of $\Pi$ if $A$ is an answer set of the aggregate reduct of $\Pi$ with respect to $A$. □

We will illustrate this definition by a number of examples.

**Example 6 (Example 4 Revisited).** Consider the grounding of program $P_4$ from Example 4

```
q(a) :- card{X:p(X,a)} = 1, r(a).
q(b) :- card{X:p(X,b)} = 1, r(b).
r(a).  r(b).  p(a,b).
```

It is easy to see that the aggregate reduct of the program with respect to any set $S$ not containing $p(a,b)$ consists of the program facts, and hence $S$ is not an answer set of $P_4$. However the program's aggregate reduct with respect to $A = \{q(b), r(a), r(b), p(a,b)\}$ consists of the program's facts and the rule

```
q(b) :- p(a,b),r(b).
```

$A$ is the answer set of the aggregate reduct, and hence $A$ is an answer set of $P_4$. □

**Example 7 (Example 5 Revisited).** Consider now the grounding

```
r :- card{X:p(X)} >= 2, q(a).
r :- card{X:p(X)} >= 2, q(b).
p(a).  p(b).  q(a).
```

of program $P_5$ from Example 5. Any answer set $S$ of this program must contain its facts. Hence $\{X : p(X) \in S\} = \{a,b\}$. $S$ satisfies the body of the first rule and must also contain $r$. Indeed, the aggregate reduct of $P_5$ with respect to $S = \{p(a), p(b), q(a), r\}$ consists of the facts of $P_5$ and the rules

```
r :- p(a),p(b),q(a).
r :- p(a),p(b),q(b).
```

Hence $S$ is the answer set of $P_5$. □

Neither of the two examples above require the application of VCP. The next example shows how this principle influences our definition of answer sets and hence our reasoning.

**Example 8 (Example 2 Revisited).** Consider program $P_0$ from Example 2

```
p(1) :- card{X : p(X)} != 1.
```

It is grounded. It has two possible answer sets, $S_1 = \{\ \}$ and $S_2 = \{p(1)\}$. The aggregate reduct of the program with respect to $S_1$ is $p(1)$. Hence, $S_1$ is not an answer set of $P_0$. The program's aggregate reduct with respect to $S_2$ is empty. So, $S_2$ is not an answer set of $P_0$ either. As expected, the program is inconsistent.

Now consider program $P_1$ from Example 2 which is also grounded:

```
p(1) :- p(0).
p(0) :- p(1).
p(1) :- card{X: p(X)} != 1.
```

Since every answer set must satisfy the first two rules of $P_1$, we only have two possible answer sets, $S_1 = \{\ \}$ and $S_2 = \{p(0), p(1)\}$. The aggregate reduct of $P_1$ with respect to $S_1$ is

```
p(1) :- p(0).
p(0) :- p(1).
p(1).
```

Its answer set, $\{p(0), p(1)\}$, is different from $S_1$, and hence $S_1$ is not an answer set of $P_1$. The aggregate reduct of $P_1$ with respect to $S_2$ is

```
p(1) :- p(0).
p(0) :- p(1).
p(1) :- p(0),p(1).
```

Its answer set is empty. So $S_2$ is not an answer set of $P_1$ either. The program is inconsistent. Similar arguments can show that the remaining programs, $P_2$ and $P_3$, from Example 2 are also inconsistent. From our standpoint this is not surprising since all these programs attempt to define $p(1)$ in terms of the totality of $p$ and hence violate VCP. □

Violation of VCP in a program rule does not necessarily render the program inconsistent. Instead it can make the rule useless.

**Example 9 (VCP and Useless Rules).** Consider a program $P_6$

```
p(1) :- card{X : p(X)} = 1.
```

The program is grounded and has two possible answer sets, $S_1 = \{\ \}$ and $S_2 = \{p(1)\}$. The aggregate reduct of $P_6$ with respect to $S_1$ is empty, $S_1$ is the answer set of the reduct and hence is an answer set of $P_6$. The aggregate reduct of $P_6$ with respect to $S_2$ consists of a useless rule

```
p(1) :- p(1)
```

and hence $S_2$ is not an answer set of $P_6$. □

All the inconsistent programs in the above examples contained rules with recursion via aggregates. Of course the definition of $p(t)$ in terms of $\{X : p(X)\}$ can involve multiple rules.

**Example 10 (Multiple Rules Aggregate Recursion).** Consider a program $P_7$

```
p(1) :- q(1).
q(1) :- card{X : p(X)} != 1.
```

It has three possible answer sets, $S_1 = \{\ \}$, $S_2 = \{p(1)\}$, and $S_3 = \{q(1), p(1)\}$. The aggregate reduct of $P_7$ with respect to $S_1$ is

```
p(1) :- q(1).
q(1).
```

The reduct with respect to $S_2$ is

```
p(1) :- q(1).
```

and the reduct with respect to $S_3$ is

```
p(1) :- q(1).
q(1) :- q(1),p(1).
```

None of the possible answer sets is an answer set of $P_7$. The program is inconsistent. □

**Discussion**: As mentioned before, there is a substantial disagreement on the intended meaning of aggregates in ASP based languages. The difficulty is related to the meaning of programs which contain recursive definition through aggregates. To understand the type of disagreements and arguments involved, let us consider program $P_2$

```
p(1) :- card{X: p(X)} >= 0
```

from Example 2. To the best of our knowledge according to all semantics of aggregates for programs which allow aggregate recursion except that of $\mathscr{A}log$ this program is consistent and has the answer set $\{p(1)\}$. The argument in favor of this goes somewhat like this: because cardinality of a set is always non-negative, $P_2$ must be “equivalent” to program

```
p(1).
```

Hence, $\{p(1)\}$ is an answer set of $P_2$.

We have two objections to this argument. First, it seems to assume the existence of the set denoted by $\{X : p(X)\}$. Otherwise, the body of the rule will be undefined and the equivalence will fail. But since $p(1)$ is defined in terms of the totality of $p$ such assumption is problematic. Of course, if we replace $P_2$ by

```
p(1) :- card{X: p(X), X != 1} >= 0
```

which avoids such a definition, the new program will have the answer set $\{p(1)\}$ in all the relevant semantics, including that of $\mathscr{A}log$.

Second, and more importantly, our objection is related to the stability principle of language design. Assuming the existence of a set denoted by $\{X : p(X)\}$, we should conclude that this set is finite and hence belongs to the domain of function *card*. Hence, intuitively, $P_2$ must be equivalent to program $P_3$ consisting of

```
p(1) :- card{X: p(X)} = Y, Y >= 0.
```

But since $P_3$ is inconsistent in all the aggregate semantics, such equivalence does not hold for a semantics in which $P_2$ is consistent. This cannot happen in $\mathscr{A}log$ because Proposition 6 in Section 6 below guarantees its stability with respect to this transformation.

Now we give a simple but practical example of a program which allows recursion through aggregates but avoids vicious circles.

**Example 11 (Defining Digital Circuits).** Consider part of a logic program formalizing propagation of binary signals through simple digital circuits. We assume that the circuit does not have a feedback, i.e., the signal coming out of the gate to its output wire cannot come back to this gate. The program may contain a simple rule

```
val(W,0) :-
      gate(G, and),
      output(W, G),
      card{W: val(W,0), input(W, G)} > 0
```

(partially) describing propagation of symbols through an *and* gate. Here $val(W,S)$ holds iff the digital signal on a wire $W$ has value $S$. Despite its recursive nature the definition of *val* avoids vicious circles. To define the signal on an output wire $W$ of an *and* gate $G$ one needs to only construct a particular subset of input wires of $G$. Since, due to the absence of feedback in our circuit, $W$ can not belong to the latter set, our definition is reasonable. To illustrate that our semantics produces the intended result, let us consider program *Circ* consisting of the above rule and a collection of facts:

```
gate(g, and).
output(w0, g).
input(w1, g).
input(w2, g).
val(w1,0).
```

The grounding of *Circ*, $ground(Circ)$, consists of the above facts and the three rules of the form

```
val(w,0) :-
      gate(g, and),
      output(w, g),
      card{W: val(W,0), input(W, g)} > 0
```

where $w$ is $w0$, $w1$, and $w2$. Let

$$S = \{gate(g,and), val(w1,0), val(w0,0), output(w0,g), input(w1,g), input(w2,g)\}.$$

The aggregate reduct of $ground(Circ)$ with respect to $S$ is the collection of facts and the rules

```
val(w,0) :-
      gate(g, and),
```

```
    output(w, g),
    input(w1, g),
    val(w1, 0).
```

where $w$ is $w0$, $w1$, and $w2$.
The answer set of the reduct is $S$ and hence $S$ is an answer set of *Circ*. As expected it is the only answer set. (Indeed it is easy to see that other possible answer sets do not satisfy our definition.) □

Our next example deals with the Company Control Problem frequently used to illustrate the power of recursive aggregates [32, 29, 33].

**Example 12 (Company Control Problem).** In [27] the problem is described as follows: "We are given a set of facts for predicate $company(X)$, denoting the companies involved, and a set of facts for predicate $ownsStk(C1,C2,Perc)$, denoting the percentage of shares of company $C2$, which is owned by company $C1$. Then, company $C1$ controls company $C2$ if the sum of the shares of $C2$ owned either directly by $C1$ or by companies, which are controlled by $C1$, is more than 50%." This problem has been encoded as the following program $P_f$

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

Intuitively, $controlsStk(C1,C2,C3,P)$ denotes that company $C1$ controls $P$ percent of $C3$ shares through company $C2$ (as $C1$ controls $C2$, and $C2$ owns $P$ percent of $C3$ shares). Predicate $controls(C1,C2)$ encodes that company C1 controls company C2." Under the semantics of $\mathscr{F}log$ and other similar languages the program, used together with the following input:

```
ownsStk (a,b,51).
ownsStk(a,c,51).
ownsStk(b,c,21).
ownsStk(c,b,21).
```

has the answer set containing $controls(a,b)$ and $controls(a,c)$, which should be the case according to the informal specification. However, if

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

is replaced by seemingly equivalent

```
#sum{P,C2 : controlsStk(C1,C2,C3,P)}=Y, Y > 50
```

the program will become inconsistent, exhibiting instability similar to that of program $P_2$ from Example 2. From the standpoint of A-log both programs are inconsistent. Indeed, $controlsStk(a,b,21,c)$ is defined in terms of the set "depending" on the truth of $controlsStk(a,c,21,b)$ and vice versa. This is a clear violation of VCP. Such a set does not exists.

It is natural to ask if the difficulty is caused by the problem itself or by its encoding? To (at least partially) answer the question we present another solution of Company Control problem which is both, stable and consistent in $\mathscr{A}log$ as well as in $\mathscr{F}log$.

To determine companies controlled by a company $A$ we consider a directed tree $T$ with the root $A$ and the links corresponding to *ownsStk* atoms from the programs "input". Procedurally, company controlled by $A$ can be found by traversing this tree level by level, at each step marking companies which are controlled by $A$. To express this idea by a logic program we introduce two relations.

$controls(A,C,N)$ which holds iff $C$ is marked as controlled by $A$ after $N$ levels of the tree had been examined. Here $N$ ranges from 0 to the number of companies.

$may\_contribute(A,B,C,N)$ which holds iff $B$ may contribute to marking $C$ as controlled by $A$ during the $N$th step of the propagation.

```
may_contribute(A,A,C,N) :- N > 0, not controls(A,C,N-1),
                           ownsStk(A,C,_).
may_contribute(A,B,C,N) :- N > 0, not controls(A,C,N-1),
                           controls(A,B,N-1),
                           ownsStk(B,C,_).
controls(A,C,N) :- N > 0, controls(A,C,N-1).
controls(A,C,N) :-
     #sum{S,B : may_contribute(A,B,C,N), ownsStk(B,C,S)} > 50.
controls(A,B) :- controls(A,B,N).
```

The program, $P_a$, is of course, still recursive, but it does not violate VCP. It can be used with any collection $I$ of atoms formed by *ownsStk*. Later we will show that $P_a \cup I$ is stratified with respect to both, aggregates and default negation and hence has an answer set. (For the definition of these terms and the corresponding theorem see Section 6.) For the input described above $P_f$ and $P_a$ produce the same answer sets. Since our paper is already rather long we decided to leave formal proof of equivalence of these programs (and possible generalization of this result) for the future work. However, thanks to Evgenii Balai, we have some experimental evidence of equivalence. Evgenii wrote a program which automatically generates an input $I$ and compares answer sets of $P_a \cup I$ and $P_f \cup I$. After running the program for several days he was not able to find a discrepancy.

Even though we showed that the Company Control problem can be solved without violating VCP principle, it is natural to try to compare the respective solutions. We believe that the latter is clearly preferable from the standpoint of teaching. The main reason, of course, is the $P_f$'s lack of stability. It is difficult to explain to a student why his program does not work, while seemingly equivalent program of his friend does. In addition, the use of parameters $N$ and $N-1$ in $P_a$ reflects the view of recursion as the method of reducing solution of a problem to solving the same problems for simpler inputs. Among other things, this may increase our confidence in correctness of our solution. One can, however, feel that the first solution is preferable since it is, in a way, closer to the original informal specification. □

So far, all our examples deal with aggregates defined on finite sets. The next two examples illustrate our definitions for aggregates whose domains contain infinite sets.

**Example 13 (Aggregates on Infinite Sets).** Consider a program $E_1$ consisting of the following rules:

```
even(0).
even(I+2) :- even(I).
q :- min{X : even(X)} = 0.
```

The program has one answer set, $S_{E_1} = \{q, even(0), even(2), \dots\}$. Indeed, the aggregate reduct of $E_1$ with respect to $S_{E_1}$ is the infinite collection of rules

```
even(0).
even(2) :- even(0).
...
q :- even(0),even(2),even(4)...
```

The last rule has the infinite body constructed in the last step of definition 4. Clearly, $S_{E_1}$ is a subset minimal collection of ground literals satisfying the rules of the reduct (i.e., its answer set). Hence $S_{E_1}$ is an answer set of $E_1$. □

**Example 14 (Programs with Undefined Aggregates).** Now consider a program $E_2$ consisting of the rules:

```
even(0).
even(I+2) :- even(I).
q :- card{X : even(X)} > 0.
```

This program has one answer set, $S_{E_2} = \{even(0), even(2), \dots\}$. Since the aggregate *card*, ranging over natural numbers, is not defined on the set $\{t : even(t) \in S_{E_2}\}$. This means that the body of the last rule is undefined. According to clause one of definition 4 this rule is removed. The aggregate reduct of $E_2$ with respect to $S_{E_2}$ is

```
even(0).
even(2) :- even(0).
even(4) :- even(2).
......
```

Hence $S_{E_2}$ is the answer set of $E_2$.[6] It is easy to check that, since every set $A$ satisfying the rules of $E_2$ must contain all even numbers, $S_{E_1}$ is the only answer set. □

## 4. Expanding $\mathscr{A}log$ by a Subset Relation

In this section, we give an informal introduction to the full version of $\mathscr{A}log$. We have already described how $\mathscr{A}log$ deals with a number of numerical functions on sets. Other numerical functions can be easily defined in a similar manner. It could also be tempting to introduce a special notation for set-theoretic operations such as union, intersection, and complement. We, however, currently believe that this is unnecessary; $p = p_1 \cup p_2$ can be easily defined by the rules:

[6] Of course this is true only because of our (somewhat arbitrary) decision to limit aggregates of $\mathscr{A}log$ to those ranging over natural numbers. We could, of course, allow aggregates mapping sets into ordinals. In this case the body of the last rule of $E_2$ will be defined and the only answer set of $E_2$ will be $S_{E_1}$.

p(X) :- $p_1$(X)
p(X) :- $p_2$(X)

and $p = p_1 \cap p_2$ can be defined by

p(X) :- $p_1$(X), $p_2$(X).

Complement $\bar{p}$ of $p$ with respect to some sort $s$ is expressed as

$\bar{p}$(X) :- s(X), not p(X).

There are, however, two important relations which still need to be added to the language – *subset* and *equality* relations between sets. We would like $\mathscr{A}log$ to be able to naturally express constructs such as *“if set A is a subset of B then ...”* and *“Let A be an arbitrary subset of B.”* Such constructs frequently appear in the language of mathematics and can also be very useful in other domains. The following is a simple non-mathematical example of the use of the first construct. The discussion below is purely intuitive. Precise syntax and semantics of the language will be defined in the next section.

**Example 15 (Subset Relation in the Rule’s Body).** Consider a knowledge base containing two complete lists of atoms:

```
taken(mike,cs1).  taken(mike,cs2).  taken(john,cs2).
required(cs1).    required(cs2).
```

The first parameter of *taken* requires the sort “student”. The other parameter ranges over the sort “classes”. Subset relation allows for a natural definition of the new relation, $ready\_to\_graduate(S)$, which holds if student $S$ has taken all the required classes from the second list:

ready_to_graduate(S) :- {C: required(C)} $\subseteq$ {C: taken(S,C)}.

The intuitive meaning of the rule is reasonably clear and corresponds to the informal specification given above. The universally quantified implication in this specification is simply replaced by the corresponding subset relation. This avoids a more complex problem of introducing universal quantifiers and some kind of implication in the rules of the language. Let $C_1$ be the program consisting of the knowledge base and the rule above.

Using our standard understanding of set-theoretic notations and the meaning of rules it is not difficult to see that the program $C_2$ consisting of $C_1$ and the closed world assumption:

```
-ready_to_graduate(S) :- not ready_to_graduate(S)
```

implies that Mike is ready to graduate while John is not. This, of course, also will be the conclusion obtained by the formally defined entailment relation.
It is worth noting that, if the list of classes taken by a student is incomplete, the closed world assumption should be removed, but the first rule still can be useful to determine people who are definitely ready to graduate. □

**Discussion**: Even though this particular story can be represented in ASP without subset relation, such representations are substantially less intuitive and less elaboration tolerant. Here is a simplified example of alternative representation suggested to the authors by one of the reviewers of a conference version of the paper.

```
ready_to_graduate(S) :- not -ready_to_graduate(S).
-ready_to_graduate(S) :- required(C), not taken(S,C).
```

(As before, $S$ ranges over students and $C$ over classes). It is easy to check that, as expected, the answer set of this new program contains $ready\_to\_graduate(mike)$ and $\neg ready\_to\_graduate(john)$. Even though in this case the answers produced by this program are also correct, the unprincipled use of default negation leads to some potential difficulties. Suppose, for instance, that a student may graduate if given a special permission, and that John succeeds in receiving such a permission from the university administration. The most natural way to express this information is by rules

```
ready_to_graduate(S) :- permitted(S).
permitted(john).
```

Unfortunately, instead of allowing John to graduate, the program becomes inconsistent. This, of course, is unintended and contradicts our intuition. No such problem exists if these two rules are added to the original representation.

The semantics of our "ready to graduate" program is fairly non-controversial since it does not contain recursion through sets and hence does not require the VCP. The next example explains an intuitive meaning of the program containing such recursion.

**Example 16 (Set atoms in The Rule Body (Use of VCP)).** Consider $P_8$

```
p(a) :- p ⊆ {X : q(X)}.
q(a).
```

where $p \subseteq \{X : q(X)\}$ stands for $\{X : p(X)\} \subseteq \{X : q(X)\}$.

The definition of $p(a)$ in $P_8$ depends on the existence of the set denoted by $\{X : p(X)\}$. In accordance with the Vicious Circle Principle, no answer set of this program can contain $p(a)$. There are only two possible answer sets of $P_8$: $S_1 = \{q(a)\}$ and $S_2 = \{q(a), p(a)\}$. $S_1$ is not an answer set since it does not satisfy the first rule. The second is ruled out by the VCP. As expected, the program is inconsistent. □

The next example illustrates a typical use of the construct "let $A$ be a subset of $B$".

**Example 17 (Set introduction rule).** Consider a simple combinatorial problem in which there is a set of children and an unlimited supply of several types of gifts. The task is to provide each child with two gifts of different types. Let us encode the problem's input using relations *child* and *gift*. A solution, assigning gifts to children, will be represented by relation *assigned*(*child*, *gift*). The $\mathscr{A}log$ solution consists of rules:

```
assigned ⊆ {C,G : child(C), gift(G)}.
:- card{G : assigned(C,G)} != 2.
:- assigned(C1,G), assigned(C2,G), C1 != C2.
```

The first rule is of the form $p \subseteq \{\bar{X} : q(X)\}$. It is read as *"let p be an arbitrary subset of q."* This is an example of a so called *set introduction rule*. It defines *assigned* as an arbitrary set of pairs matching children with gifts. The second rule is a constraint which guarantees that every child is assigned exactly two different gifts. □

**Discussion**: This type of generate and test programs are very typical for ASP. The generate part is normally encoded by a disjunction

```
assigned(C,G) or -assigned(C,G)
```

or by a choice rule (e.g., in SMODELS [4])

```
{assigned(C,G): child(C), gift(G)}.
```

We believe that although the disjunctive rule is perfectly understandable after some explanation, it has a disadvantage of not having a clear analogue in mathematics or natural language. Moreover, it requires introduction of $\neg assigned(C,G)$ which does not seem to be warranted by the problem.

On another hand, the choice rule $\{p(\bar{X}) : q(\bar{X})\} \leftarrow body$ of [4] implemented in CLINGO and other similar systems may look very similar to set introduction rule. The rule is usually understood as a non-deterministic choice which allows the reasoner to include in an answer set $S$ of the program an arbitrary collection of atoms of the form $p(t)$ such that $q(t) \in S$. The two constructs, however, have different meanings.

**Example 18 (Set Introduction versus Choice Rule).** Consider, for instance, a program $P_{10}$

```
q1(0).    q1(1).
q2(0).    q2(2).
p ⊆ {X: q1(X)}.
p ⊆ {X: q2(X)}.
```

According to our intuitive reading of the rules, the program defines $p$ as an arbitrary subset of the intersection of $q1$ and $q2$. As a result $P_{10}$ has two answer sets: $S_1 = \{q1(0), q1(1), q2(0), q2(2)\}$ and $S_2 = S_1 \cup \{\, p(0)\}$. The corresponding program of CLINGO

```
q1(0).    q1(1).
q2(0).    q2(2).
{p(X): q1(X)}.
{p(X): q2(X)}.
```

treats $p$ as an arbitrary subset of the union of $q1$ and $q2$ and consequently has other answer sets including, say, $S_1 \cup \{p(1)\}, S_1 \cup \{p(1), p(2)\}$, etc. This may suggest that program $P_{10}$ of $\mathscr{A}log$ can be modeled in CLINGO by replacing $p$ in the rules above by $p1$ and $p2$ and defining $p$ by the rule `p(X) :- p1(X), p2(X).` Even though the transformation works in this case it is not sound in general. □

The following example shows the difficulty of generalizing this transformation which is, of course, related to the VCP.

**Example 19 (Set Introduction versus Choice Rule (continued)).** Consider a program $P_{11}$

```
q(1).
p ⊆ {X : q(X)}.
q(2) :- p(1).
```

The program defines $p$ in terms of the totality of $q$ which, in turn, is defined in terms of $p$. This clearly violates VCP. The second rule is useless and the only answer set of the program is $\{q(1)\}$. Replacing the set introduction rule of $P_{11}$ by the choice rule leads to the CLINGO program

```
q(1).
{p(X) : q(X)}.
q(2) :- p(1).
```

which has three answer sets: $S_1 = \{q(1)\}$, $S_2 = \{q(1), q(2), p(1)\}$, and $S_3 = \{q(1), q(2), p(1), p(2)\}$.

Even though after the result is obtained this does not look unreasonable, we did not have sufficiently developed intuition to predict the program's behavior. □

Overall, we prefer the set introduction rule to both "generating" constructs discussed above. We believe that it has more intuitive reading (after all everyone is familiar with the statement "let $p$ be an arbitrary subset of $q$"), while explanation of a choice rule in terms of "generation" has more procedural flavor. Moreover, in the next section we hope to demonstrate the relative simplicity of the definition of its formal semantics as compared with that of the choice rule.

## 5. Syntax and Semantic of $\mathscr{A}log$

### 5.1. Syntax

As before, we assume a fixed signature $\Sigma$ with a finite collection of predicate and function symbols and (possibly infinite) collection of object constants together with a finite collection $\mathscr{A}$ of aggregate names. Terms and literals over signature $\Sigma$ are referred to as *regular*. To define syntax of the full language, we first expand the notion of a set atom defined for the aggregate programs in 3.1.

**Definition 6 (Set Atoms of $\mathscr{A}log$).** Let $f$ be an aggregate name, $S$, $S_1$, $S_2$ be set expressions, $k$ be a natural number, $\odot$ be an arithmetic relation $>, \geq, <, \leq, =$ or !=, $\otimes$ be $\subset, \subseteq$, or $=$ of sets, and $p$ be a predicate symbol.

A *set atom* of $\mathscr{A}log$ is an expression in one of the following forms

$$f(S) \odot k \tag{4}$$

$$S_1 \otimes S_2 \tag{5}$$

$$p \otimes S \tag{6}$$

$$S \otimes p \tag{7}$$

*The atoms of the form (4) will be referred to as* aggregate atoms. □

$f_1(S_1) \odot f_2(S_2)$ may be used as an abbreviation for $f_1(S_1) = Y_1, f_2(S_2) = Y_2, Y_1 \odot Y_2$. Set atoms and regular atoms (literals) over signature $\Sigma$ are referred to as $\Sigma$-*atoms* ( $\Sigma$-*literals*). Regular and set atoms are referred to as *atoms*.

**Definition 7 (Rules and Programs of $\mathscr{A}log$).** A *rule* of $\mathscr{A}log$ is an expression of the form

$$head \leftarrow body \tag{8}$$

where *head* is, a (possibly infinite) disjunction of regular literals, or a set atom of the form $p \subseteq S$, $S \subseteq p$, or $p = S$, and *body* is a (possibly infinite) collection of regular literals (which may be preceded by *not*) and set atoms. We call *head* the *head* of the rule, and *body* the *body* of the rule.

A rule is called a *set introduction rule for p* if its head is a set atom. It is called a *constraint* if its head is empty. A rule is called a *proper disjunctive rule* if it is neither a set introduction rule nor a constraint.

A *program* of $\mathscr{A}log$ is a collection of $\mathscr{A}log$'s rules. □

Finally, let us notice that the definition of bound and free variables and that of grounding for arbitrary $\mathscr{A}log$ programs are the same as for aggregate programs.

*5.2. Semantics*

The semantics of a ground program $\Pi$ of $\mathscr{A}log$ will be given in two steps. First, we define the semantics for programs without set introduction rules. In the second step *set introduction reduct* will be used to define the semantics for arbitrary programs. Satisfiability of aggregate atoms is given in Definition 3. Satisfiability of non-aggregate set atoms by a set $A$ of ground regular literals is defined as expected, e.g., $p \otimes \{X : q(X)\}$ is *satisfied* by $A$ if $\{t : p(t) \in A\} \otimes \{t : q(t) \in A\}$. Similarly for the remaining set atoms.

*5.2.1. Programs without Set Introduction Rules*

The definition of an answer set for a ground program $\Pi$ not containing set introduction rules requires a very small change in the definition of an aggregate reduct from section 3. We just need to add the additional clause explaining the meaning of occurrences of atoms $p \otimes S$ and $S \otimes p$ in the bodies of program rules. Since the change is small and relations $\otimes$ can be viewed as aggregates defined on pairs of sets we will retain the original name of the reduct.

**Definition 8 (Aggregate Reduct for Programs without Set Introduction Rules).** The *aggregate reduct* of $\Pi$ with respect to a set $A$ of ground regular literals is obtained from $\Pi$ as follows:

1. replace all occurrences of atoms of the form $p \otimes S$ and $S \otimes p$ in $\Pi$ by $\{\bar{X} : p(\bar{X})\} \otimes S$ and $S \otimes \{\bar{X} : p(\bar{X})\}$ respectively,
2. remove rules containing set atoms which are *false* or *undefined* in $A$,
3. for every occurrence of a set expression $\{X : cond(X)\}$ in a rule add to the body of the rule all atoms of the form $cond(t)$ such that $cond(t)$ is true in $A$, and
4. remove from the rules all their set atoms.

□

The definition of an answer set of $\Pi$ remains unchanged.

Let us illustrate the definition by showing that program $P_8$ from Example 16 is indeed inconsistent.

**Example 20 (Example 16 Revisited).** We repeat $P_8$ here:

```
p(a) :- p ⊆ {X : q(X)}.
q(a).
```

In Example 16, we gave an informal argument showing that the program violates VCP and is inconsistent. Here is a formal argument establishing its inconsistency. To shorten the discussion we use the supportedness property of $\mathscr{A}log$ proven in Section 6. By this property, there are only two possible answer sets of $P_8$: $S_1 = \{q(a)\}$ and $S_2 = \{q(a), p(a)\}$. To produce the aggregate reduct of $P_8$ with respect to $S_1$ we replace the first rule by

```
p(a) :- {X:p(X)} ⊆ {X : q(X)}.
```

and replace the set atom by $q(a)$. The aggregate reduct is

```
p(a) :- q(a).    q(a).
```

The aggregate reduct of $P_8$ with respect to $S_2$ is

```
p(a) :- p(a),q(a).    q(a).
```

Clearly, $S_1$ does not satisfy the rules of its reduct and hence is not an answer set of $P_8$; $S_2$ does satisfy the rules of the second reduct but so is its proper subset $\{q(a)\}$. So, $S_2$ is not an answer set of $P_8$ either. □

It is not difficult to also check that our formal definition justifies informal arguments from Example 15.

*5.2.2. Programs with Set Introduction Rules*

A set introduction rule with head $p \subseteq S$ (where $p$ is a predicate symbol and $S$ is a set expression) defines set $p$ as an arbitrary subset of $S$; a rule with head $p = S$ simply gives $S$ a different name; $S \subseteq p$ defines $p$ as an arbitrary superset of $S$.

The formal definition of answer sets of programs with set introduction rules is given via a notion of *set introduction reduct*.

**Definition 9 (Set Introduction Reduct).** The *set introduction reduct* of a ground $\mathscr{A}log$ program $\Pi$ with respect to a set $A$ of ground regular literals is obtained from $\Pi$ by

- replacing every set introduction rule of $\Pi$ whose head is not true in $A$ by

$$\leftarrow body.$$

- replacing every set introduction rule of $\Pi$ whose head $p \subseteq \{\bar{X} : q(\bar{X})\}$ (or $p = \{\bar{X} : q(\bar{X})\}$ or $\{\bar{X} : q(\bar{X})\} \subseteq p$) is true in $A$ by

$$p(\bar{t}) \leftarrow body, A_q$$

  where $A_q = \{q(\bar{t}) : q(\bar{t}) \in A\}$ for each $p(\bar{t}) \in A$.

(The definition is similar to that presented in [28] and [34]. The new element is the formalization of VCP by the introduction of $A_q$ in the second rule).

Set $A$ is an *answer set* of $\Pi$ if it is an answer set of the set introduction reduct of $\Pi$ with respect to $A$. □

Let us illustrate this definition by the following example.

**Example 21 (Set Introduction Rule).** Consider program $P_{12}$

```
q(a).
p ⊆ {X:q(X)}.
```

Intuitively, the program has answer sets $A_1 = \{q(a)\}$ where the set $p$ is empty and $A_2 = \{q(a), p(a)\}$ where $p = \{a\}$. Formally, the set introduction reduct of $P_{12}$ with respect to $A_1$ is

```
q(a)
```

and hence $A_1$ is an answer set of $P_{12}$. The reduct of $P_{12}$ with respect to $A_2$ is

```
q(a).
p(a) :- q(a).
```

and hence $A_2$ is also an answer set of $P_{12}$. It is easy to check that there are no other answer sets.

Next, recall program $P_{10}$

```
q1(0).     q1(1).
q2(0).     q2(2).
p ⊆ {X : q1(X)}.
p ⊆ {X : q2(X)}.
```

from Example 18 and let *Facts* be the set of facts of the program. It is easy to see that the set introduction reduct of $P_{10}$ with respect to, say, $S_2 = Facts \cup \{p(0)\}$ is

*Facts*

```
p(0) :- q1(0), q1(1).
p(0) :- q2(0), q2(2).
```

and hence $S_2$ is an answer set of $P_{10}$. Similarly for $S_1 = Facts$. Consider now $S = Facts \cup \{p(1)\}$. The reduct of $P_{10}$ with respect to $S$ consists of *Facts*, rule,

```
p(1) :- q1(0), q1(1).
```

and constraint

```
:-
```

with the empty head and the empty body. Clearly, it cannot be satisfied, and hence $S$ is not an answer set of $P_{10}$.

Finally, recall program $P_{11}$

```
q(1).
p ⊆ {X : q(X)}.
q(2) :- p(1).
```

from Example 19, which contains recursion through aggregates. The reduct of the program with respect to $S_1 = \{q(1)\}$ is

```
q(1).
q(2) :- p(1).
```

and hence $S_1$ is an answer set of $P_{11}$. The reduct of $P_{11}$ with respect to $S_2 = \{q(1), q(2), p(1), p(2)\}$ is

```
q(1).
p(1) :- q(1), q(2).
p(2) :- q(1), q(2).
q(2) :- p(1).
```

The set introduction rule of $P_{11}$ is useless and $S_2$ is not an answer set of $P_{11}$. The example formally shows that transformation from $\mathscr{A}log$ to CLINGO described in Example 19 is not sound. □

We conclude the section with an additional example of the use of subset introduction rule for knowledge representation. This time the rule will be used to define synonyms.

**Example 22 (Synonyms).** Suppose we have a set of cars identified (for simplicity) by their owners. The set can be represented by the program $D_1$ consisting of atoms, say,

```
car(bob).
car(mary).
```

To check if his car is in the list, an English speaking user Bob will simply pose a query $?car(bob)$. Suppose now we would like to make the database available to Spanish speaking people by allowing them to ask a query $?carro(name)$. One natural way to allow this would be to consider program $D_2$ obtained by expanding $D_1$ by a rule:

```
carro(X) :- car(X).
```

This is a reasonable solution but it does not protect us from difficulties related to accidental addition of, say, $carro(jose)$ to $D_1$. Our intent was to define *carro* as a synonym for *car*, so that English and Spanish speaking people will be guaranteed to get the same answers. Hence, such an accidental addition shall not be allowed. Our solution does not guarantee this. If, however, we add another constraint

```
:- carro(X), not car(X)
```

the program with $carro(jose)$ will, as expected, become inconsistent.

Here is an alternative solution which uses subset introduction rule with equality: let $D_3$ be obtained from $D_1$ by adding to it rule

```
carro = {X:car(X)}.
```

Clearly, *car* and *carro* are synonyms. It is easy to check that $D_3$ has one answer set, $\{car(bob), car(mary), carro(bob), carro(mary)\}$ and hence queries $carro(bob)$, $car(bob)$, etc., will be answered correctly. However, the expansion of $D_3$ by $carro(jose)$ will cause inconsistency. □

## 6. Properties of $\mathscr{A}log$ Programs

In our principles of language design we suggested that a new language should come with mathematical theory facilitating its use for knowledge representation and programming. In this section we give some important properties of $\mathscr{A}log$ programs, contributing to the development of such theory. Propositions 1 and 2 ensure that, as in regular ASP, answer sets of $\mathscr{A}log$ programs are formed using the program rules together with the rationality principle. Proposition 3 is the $\mathscr{A}log$ version of the Splitting Set Theorem – basic technical tool used for computing answer sets and for theoretical investigations of ASP and its extensions [35, 36, 37]. Proposition 4 states that the stratified programs are consistent. Proposition 6 shows an example of the stability of $\mathscr{A}log$ under some equivalent arithmetic transformation, and Proposition 7 and 8 give results on the complexity of $\mathscr{A}log$ programs.

### 6.1. Basic Properties

**Proposition 1 (Rule Satisfaction and Supportedness).** *Let A be an answer set of a ground program* $\Pi$ *of* $\mathscr{A}log$*. Then*

- *A satisfies every rule r of* $\Pi$.
- *If* $p(\bar{t}) \in A$ *then there is a rule r from* $\Pi$ *such that the body of r is satisfied by A and*

- *r is a proper disjunctive rule and $p(\bar{t})$ is the only atom in the head of r which is true in A or*
- *r is a set introduction rule defining p.*

*(In both cases it is often said that r supports $p(\bar{t})$).*

Proposition 1 extends similar result for disjunctive logic programs from [38].

By the intuitive and formal meaning of set introduction rules, the anti-chain property no longer holds for arbitrary programs of $\mathscr{A}log$. But it remains to be true for programs without set introduction rules.

**Proposition 2 (Anti-chain Property).** *If $\Pi$ is a program without set introduction rules then there are no $\mathscr{A}log$ answer sets $A_1$, $A_2$ of $\Pi$ such that $A_1 \subset A_2$.*

### *6.2. Splitting an $\mathscr{A}log$ Program*

In this section we present an $\mathscr{A}log$ analogue of splitting set theorem [39, 36, 37]. Since ground $\mathscr{A}log$ program contains variables, the definition of splitting set is slightly more involved than usual. We will need auxiliary notions of "potential support" and "set expression determined by a signature."

**Definition 10 (Potential Support).** A rule $r$ of a ground program $\Pi$ is a *potential support* for a regular literal $l$ if the head of $r$ is a disjunction containing $l$ or $r$ is a set introduction rule defining $p$ and $l = p(\bar{t})$. □

For example, in a program

```
q(0) :- not s(0).
p ⊆ {X : q(X)}.
```

the first and second rules are potential supports for $q(0)$ and $p(0)$ respectively; $s(0)$ has no potential support in the program.

**Definition 11 (Set Expressions Determined by Sets of Literals).** Let $\Pi$ be a ground program with signature $\Sigma$. We say that *the value of a set expression $\{\bar{X} : cond\}$ of $\Sigma$ is determined by a set S of user-defined literals* if for any consistent[7] ground instance $cond(\bar{t})$ in $\Sigma$, either some user-defined literal in $cond(\bar{t})$ has no potential support in $\Pi$ or every user-defined literal of $cond(\bar{t})$ is in $S$. □

[7] Recall that a collection of ground literals is consistent if it has a model.

Let $\Pi$ be a program with signature $\Sigma$ consisting of predicate symbols $p$ and $q$, object constants 0, 1, 2 and rules

```
q(0) :- card{X : p(X), X ≠ 1} > 0.
p(0).
```

It is easy to check that the value of $\{X : p(X), X \neq 1\}$ is determined by the set of literals $S = \{p(0)\}$. Indeed, there are two consistent ground instances of the set condition: $\{p(0), 0 \neq 1\}$ and $\{p(2), 2 \neq 1\}$. The only user-defined literal in the first condition is from $S$; although $p(2)$ in the second condition is not from $S$, it has no potential support in the program.

Now we are ready for the main definition.

**Definition 12 (Splitting Set).** Let $\Pi$ be a ground $\mathscr{A}log$ program with signature $\Sigma$.

A set $S$ of ground user-defined literals is called a *splitting set* of $\Pi$ if

- If $r$ is a potential support of $l \in S$ then every user-defined literal belonging to $r$ is in $S$ and the value of every set expression occurring in $r$ is determined by $S$.
- If $r$ is a set introduction rule for $p$ and $p(\bar{t}_0) \in S$ for some $\bar{t}_0$ then $p(\bar{t}) \in S$ for every (properly typed) $\bar{t}$ of $\Sigma$.

A splitting set $S$ of $\Pi$ splits the program into two parts: the *bottom* of $\Pi$ relative to $S$ consisting of all potential supports of literals from $S$, and the remaining part of $\Pi$ called the *top* of $\Pi$ relative to $S$. □

**Example 23 (Splitting Set).**
(a) **The Circuit**: Consider a sorted signature $\Sigma$ with object constants $w0, w1, w2, w3$ for *wires*, $g1$ and $g2$ for *gates* and 0 and 1 for *signals*, and predicates $val(wire, signal)$ and $input(wire, gate)$. Let program $E_1$ consist of rules:

```
val(w0,0) :- card{W: val(W,0), input(W, g1)} > 0.
val(w3,0) :- card{W: val(W,0), input(W, g2)} > 0.
```

and $\Sigma_0$ be a signature obtained from $\Sigma$ by dropping constants $w3$ and $g2$. Let us check that the set $S$ of all atoms of $\Sigma_0$ is a splitting set of $E_1$. To do that it is sufficient to check that value of $\{W : val(W,0), input(W, g1)\}$ is determined by $S$. This is true, since the only ground instance of the corresponding condition which is not formed by atoms of $S$ is $\{val(w3,0), input(w3,g1)\}$ and $input(w3,g1)$ has no potential support in $E_1$. The set splits the program into the bottom consisting of the first rule, and the top consisting of the second one.

(b) **Role of Consistency Condition**: The next program $E_2$, consisting of rules:

```
p(a) :- q(b).
q(b) :- card{X: p(X),X != a} = 0.
```

illustrates the use of consistency condition in Definition 11. Let us show that $S = \{q(b), p(b)\}$ is a splitting set of $E_2$. To do that we need to show that the value of expression $\{X : p(X), X \neq a\}$ is determined by $S$. The only consistent ground instance of condition $\{p(X), X \neq a\}$ is $\{p(b), b \neq a\}$ and its only user-defined literal $p(b)$ is in $S$. Hence, Definition 11 is satisfied. Note, that if we were to remove consistency condition from Definition 11 $S$ would not be a splitting set of $E_2$.

(c) **Program with Set Introduction Rule**: Consider a program $E_3$ with rules:

```
s :- not p(1).
p ⊆ {X: q(X)}.
q(1).        q(2).
```

and signature $\Sigma$ implicitly defined by these rules. Let $\Sigma_0$ be obtained from $\Sigma$ by dropping $s$ and show that $S$ consisting of atoms of $\Sigma_0$ is a splitting set of $E_3$. This is the case, since both ground instances of $p$ are in $S$ and both ground instances of $q$ are in $S$. Note that $S_1 = S \setminus \{p(1)\}$ is not a splitting set of $E_3$, since it violates the second condition of Definition 12. □

Now we are ready to formulate Splitting Set Theorem for $\mathscr{A}log$.

**Proposition 3 (Splitting Set Theorem).** *Let* $\Pi$ *be a ground* $\mathscr{A}log$ *program,* $S$ *be a splitting set of* $\Pi$*, and* $\Pi_1$ *and* $\Pi_2$ *be the bottom and the top of* $\Pi$ *relative to* $S$ *respectively. Then a set* $A$ *is an answer set of* $\Pi$ *iff* $A \cap S$ *is an answer set of* $\Pi_1$ *and* $A$ *is an answer set of* $(A \cap S) \cup \Pi_2$.

Instead of the formulation of Splitting Set Theorem given above it is sometimes convenient to use the following Corollary. First, some definitions.

Let $\Pi$, $S$, $\Pi_1$ and $\Pi_2$ be as in the theorem above, and let $B$ be an answer set of $\Pi_1$. By $Red(\Pi_2, B)$ we denote the program obtained from $\Pi_2$ by

- removing from $\Pi_2$ every rule whose body contains $l \in S$ such that $l \notin B$ or contains *not* $l$ such that $l \in B$, and
- removing all remaining extended literals formed from elements of $S$ from the rules of $\Pi_2$.

**Corollary 1.** *$A$ is an answer set of $\Pi$ iff $A \cap S$ is an answer set of $\Pi_1$ and $A$ is an answer set of $(A \cap S) \cup Red(\Pi_2, A \cap S)$.*

The Corollary follows immediately from the Splitting Set Theorem and the definition of answer sets.

### *6.3. Stratification of $\mathscr{A}$-log Programs*

In this section, we define a notion of stratified $\mathscr{A}log$ program and show that every such program is consistent, i.e., has an answer set. To achieve consistency we prohibit programs with classical negation and constraints, require a program to be stratified with respect to default negation (see [40]), and impose an additional condition of stratification with respect to sets. The latter divides the program into levels and ensures that a set $p$ or its instance can be defined in terms of a set $q$ only if the membership in $q$ has already been fully determined on the previous levels. Similar idea in the context of $\mathscr{F}log$ was suggested in [27], but there authors were mainly interested in complexity of stratified programs and not in their consistency. There are substantial technical differences between the two approaches which will be discussed at the end of the section.

#### *6.3.1. Definition of Stratification and a Consistency Result*

By *leveling* $\| \ \|_\lambda$ for $\Pi$ we mean a mapping from ground regular literals of $\Pi$ onto the collection of ordinals from 0 to some (recursive) ordinal $\lambda$.

**Definition 13 (Stratification of $\mathscr{A}log$ Programs).**
Let $\Pi$ be a ground program of $\mathscr{A}log$.

1. A leveling $\| \ \|$ *stratifies* $\Pi$ *with respect to sets* if for every rule $r$ of $\Pi$ and every regular literal $l$ potentially supported by $r$:
   (a) for every $l_i$ potentially supported by $r$, $\| l \| = \| l_i \|$, and
   (b) every set expression occurring in $r$ is determined by set $S_l = \{l_i \ : \| l \| > \| l_i \|\}$.
2. $\| \ \|$ *stratifies* $\Pi$ *with respect to default negation* if for every rule $r$ of $\Pi$ and every regular literal $l$ potentially supported by $r$,

(a) $\| l \| > \| l_k \|$ if *not* $l_k$ is an element of the body of $r$,

(b) $\| l \| \geq \| l_k \|$ if $l_k$ is an element of the body of $r$.

An $\mathscr{A}log$ program $\Pi$ is called *stratified* if

1. All rules of $\Pi$ are finite.
2. $\Pi$ contains no constraints and no classical negation $\neg$.
3. $\Pi$ has at most one set introduction rule for every $p$.
4. If $\Pi$ contains a set introduction rule for $p$ then no atom of the form $p(\bar{t})$ occurs in the heads of proper disjunctive rules of $\Pi$.
5. Some leveling $\| \; \|$ stratifies $\Pi$ with respect to both, sets and default negation.

**Proposition 4 (Consistency of Stratified Programs).** *A stratified program* $\Pi$ *of* $\mathscr{A}log$ *is consistent.*

For example, it is easy to check that the program

```
p(0) :- card{X : q(X)} ≥ 0
```

and the program

```
p ⊆ {X : q(X)}
```

are stratified by a leveling $\| q(0) \| = 0$ and $\| p(0) \| = 1$.

The program $C_1$ from Example 15 is stratified by the leveling assigning 0 to atoms formed by *taken* and *required* and 1 to those formed by *ready_to_graduate*.

In all these cases this could have been proven by using a weaker form of condition 1b of the definition of stratification. We could simply require the set expressions occurring in rules of level $\alpha$ have predicate symbols fully defined on the previous levels. This is not the case in the following example.

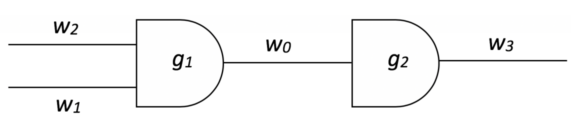


Figure 1: A circuit

**Example 24 (Stratification).** Consider an electrical circuit from Fig 1 and a program $E_4$ consisting of the circuit's description

```
input(w1, g1).   input(w2, g1).   input(w0, g2).
output(w0, g1).  output(w3, g2).
gate(g1, and).   gate(g2, and).
```

input signals

```
val(w1,0).       val(w2,1).
```

and rules

```
val(w0,0) :- card{W: val(W,0), input(W, g1)} > 0.
val(w3,0) :- card{W: val(W,0), input(W, g2)} > 0.
```

It is not difficult to check that the program is stratified by a leveling $\|\ \|$ such that for every signal $s$, wire $w$ and gate $g$:

$\| gate(g,and) \| = 0$
$\| input(w,g) \| = \| output(w,g) \| = 0$
$\| val(w,s) \| = 0$ if $w$ is $w1$ or $w2$.
$\| val(w0,s) \| = 1$
$\| val(w3,s) \| = 2$

For that, it is sufficient to check that $\|\ \|$ stratifies $E_4$ with respect to sets, i.e., the two rules of $E_4$ satisfy condition (1b) from the definition of stratification. To show this for the first rule let $l = val(w0,0)$ and $A = \{W : val(W,0), input(W,g1)\}$. Then $S_l$ consists of all atoms of level 0. Let $I = \{val(w,0), input(w,g1)\}$ be an instance of $A$. If $w$ is different from $w1$ and from $w2$ then $input(w,g1)$ has no potential support in $E_4$. Otherwise, $I \subset S_l$. Thus $A$ is determined by $S_l$. Similarly, for the second rule, and therefore $E_4$ is stratified.

One can also easily establish that programs $E_2$ and $E_3$ from Example 23 are stratified, while programs which violate the VCP, such as $P_0$–$P_3$ from Example 2, are not. It is also not difficult to show that program $P_a \cup I$ from the company control Example 12 is stratified by the leveling:
$\| ownsStk(\_,\_,\_) \| = 0.$
$\| controls(\_,\_,0) \| = 0.$
$\| may\_contribute(\_,\_,\_,0) \| = 0.$
For every $0 < k \leq n$, where $n$ is the number of companies,
$\| may\_contribute(\_,\_,\_,k) \| = 2k$, and
$\| controls(\_,, k) \| = 2k+1.$
Finally, $\| controls(\_,\_) \| = \| controls(\_,\_,n) \|$ .
and is therefore consistent by Proposition 4.

□

Next we give an example of a useful non-stratified program and show how the Splitting Set Theorem can be used to reduce it to an equivalent stratified one.

**Example 25 (Full Digital Circuits).** Let $E_5$ be a program consisting of the facts from $E_4$ (describing the circuit in Fig 1) and a rule:

```
val(W,0) :-
       output(W,G),
       gate(G,and),
       card{W: val(W,0), input(W, G)} > 0.
```

Let us show that $E_5$ is not stratified.

Suppose there is a leveling $\| \ \|$ which stratifies $E_5$ with respect to sets. Consider three cases:

(1) $\| \mathit{val}(w1,0) \| = \| \mathit{val}(w2,0) \|$.

Notice that the grounding of $E_5$, $\mathit{ground}(E_5)$, contains rules

```
(a) val(w1,0) :-
        output(w1,g1),
        gate(g1,and),
        card{W: val(W,0), input(W, g1)} > 0.
(b) val(w2,0) :-
        output(w2,g1),
        gate(g1,and),
        card{W: val(W,0), input(W, g1)} > 0.
```

Atom $\mathit{val}(w1,0)$ is potentially supported by rule (a), and hence, by the definition of stratification, the value of set $A = \{W : \mathit{val}(W,0), \mathit{input}(W,g1)\}$ occurring in the rule should be determined by $S_{\mathit{val}(w1,0)}$ consisting of atoms with levels lower than that of $\mathit{val}(w1,0)$. Let $\{\mathit{val}(w2,0), \mathit{input}(w2,g1)\}$ be an instance of $A$. By (1), $\mathit{val}(w2,0) \notin S_{\mathit{val}(w1,0)}$, but it is potentially supported by rule (b). Hence, $A$ is not determined by $S_{\mathit{val}(w1,0)}$, and (1) is impossible. Suppose

(2) $\| \mathit{val}(w1,0) \| > \| \mathit{val}(w2,0) \|$.

But this is impossible too since $\mathit{val}(w2,0)$ is potentially supported by rule (b) but the set expression occurring in the rule is not determined by $S_{\mathit{val}(w2,0)}$ which does not contain $\mathit{val}(w1,0)$.

A symmetry between rules (a) and (b) imply impossibility of

(3) $\| val(w1,0) \| < \| val(w2,0) \|$.

Hence, $E_5$ is not stratified.

Even though we can not prove consistency of $E_5$ directly by Proposition 4, this can be easily done by combining this proposition with the splitting set theorem.

This will allow us to eliminate rules (a) and (b), which contain unsupported atoms $output(w1,g1)$ and $output(w2,g1)$ from the program. To do that first notice that the set $S$ of ground atoms formed by $input$, $output$ and $gate$ is a splitting set of $ground(E_5)$. Let $B$ be the bottom of the program and $T$ be its top. By definition, $Red(T,B)$ removes from $T$ (a) all the rules containing atoms from $S$ which are not in $B$ and (b) all remaining occurrences of atoms from $S$. It is not difficult to see that $B \cup Red(T,B)$ is exactly the program $E_4$ from the previous example, where it was shown to be stratified and thus consistent. Hence, by Corollary 1, so is $E_5$.

This method of using the splitting set theorem to remove useless atoms and rules which prevent a program from being stratified provide a powerful tool for proving consistency of $\mathscr{A}log$ programs. □

*6.3.2. Discussion of the Definition of Stratification*

While some restrictions in Definition 13 are inherited from the notion of stratified programs of Answer Set Prolog, others are pertinent to the new features of $\mathscr{A}log$. In what follows we briefly explain these restrictions.

1. **Prohibition of infinite rules**. The following example shows that if infinite rules are allowed even a positive disjunctive program[8] may not have an answer set. Since stratification is a consistency condition, the prohibition is justified.

**Example 26.** Let $p$ be a predicate defined on the set of natural numbers and let $\Pi$ consist of rules:

```
p(0) or p(1) or p(2) ... .
p(1) or p(2) or p(3) ... .
p(2) or p(3) or p(4) ... .
...
```

Let us show that the program has no answer set. Suppose $A$ is an answer set of $\Pi$. Since $A$ must satisfy all the rules of $\Pi$ it cannot be empty. Hence, there is $k$ such that $p(k) \in A$. Note, that by construction, rule $k+1$ of $\Pi$ is of the form

[8] Recall that a program is called *positive* if for every rule of this program, its head is a non-empty disjunction of regular atoms and its body is a collection of regular atoms.

$p(k+1)$ or $p(k+2)\ldots.$

Since this rule is also satisfied by $A$ there must be $m > k$ such that $p(m) \in A$. This means that $A \setminus \{p(k)\}$ also satisfies the rules of $\Pi$ which contradicts our assumption. Hence, $\Pi$ is inconsistent. □

2. **Restrictions on the set introduction rules**. The next example shows that simply removing these restrictions leads to inconsistency.

**Example 27.** Consider program $\Pi_1$

```
q2(1).
p ⊆ {X:q1(X)}
{X:q2(X)} ⊆ p
```

According to the second rule $p$ must be empty, but, by the first and third rules it must contain 1, i.e., $\Pi_1$ is inconsistent.

Similarly, $\Pi_2$:

```
p ⊆ {X:q(X)}
p(1).
```

□

It is worth noticing that the prohibition of multiple set introduction rules for the same set $p$ is not as severe as it may seem at the first glance. In many cases such multiple rules can be replaced by a single one. For instance, a program $\Pi_3$ containing set introduction rules defining $p$ in terms of $\{X : q1(X)\}$ and $\{X : q2(X)\}$ can be replaced by the program $\Pi_4$ obtained from $\Pi_3$ by removing all such rules and adding rules

```
q(X) :- q1(X), q2(X).
p ⊆ {X:q(X)}.
```

where $q$ is a new predicate symbol. Similarly, if $p$ were defined as a superset of both, $\{X : q1(X)\}$ and $\{X : q2(X)\}$ the corresponding set introduction rules could have been replaced by

```
q(X) :- q1(X).
q(X) :- q2(X).
{X:q(X)} ⊆ p.
```

So the restriction on the number of set introduction rules can be relaxed but, for simplicity, we will not do it here.

### 6.3.3. Stratifications in $\mathscr{A}log$ and $\mathscr{F}log$ – a Comparison

To the best of our knowledge, the notion of stratification for programs with aggregates was first introduced in [27] for programs of $\mathscr{F}log$ in the context of studying complexity of logic programs. Comparison between the two definitions is not entirely trivial since syntactically the programs of $\mathscr{F}log$ are not necessarily programs of $\mathscr{A}log$. However, for an aggregate program $\Pi$ without infinite rules, this is not the case. Syntactically, $\Pi$ can be viewed as a program of $\mathscr{A}log$ as well as $\mathscr{F}log$. Let us denote $\Pi$ under $\mathscr{A}log$ semantics by $\Pi^A$ and under $\mathscr{F}log$ semantics by $\Pi^F$. This takes care of the syntactic difficulty. Since the notion of stratification introduced in our paper is different from that in [27], we separate between them by using terms *A-stratification* and *F-stratification*. Let us briefly describe the relationship between these two notions.

By examining the definitions of $A$-stratification and $F$-stratification it is easy to check that

(a) If $\Pi^F$ is $F$-stratified then $\Pi^A$ is $A$-stratified.

(b) The opposite is not true. For instance program $E_4$ in Example 24 is $A$-stratified but not $F$-stratified.

It is known that every $\mathscr{A}log$ answer set of $\Pi^A$ is also an $\mathscr{F}log$ answer set of $\Pi^F$ and that, in general, the opposite is not true [41, 42]. However, it is not difficult to prove that for a broad class of programs $\Pi^A$ and $\Pi^F$ have the same answer sets. To make it precise, we need the following definition.

**Definition 14 (Compatible Programs).** *A ground aggregate program* $\Pi$ *is called* $\mathscr{A}\mathscr{F}$-compatible *if it contains no infinite rules.* □

**Proposition 5 ( $\mathscr{A}log$ vs $\mathscr{F}log$ Semantics under F-stratification).** *If an* $\mathscr{A}\mathscr{F}$*-compatible program* $\Pi$ *is F-stratified, then A is an* $\mathscr{A}log$ *answer set of* $\Pi$ *iff it is an* $\mathscr{F}log$ *answer set of* $\Pi$.

The proof of a variant of this proposition, together with additional results, was first introduced in [43]).

There are other questions about stratification left unanswered in this paper, but they will be a subject of further investigation.

### 6.4. Stability Condition

The next theorem shows that the stability condition discussed earlier with respect to program $P_2$ in section 3.2 holds for an arbitrary $\mathscr{A}log$ program. (Recall that it is not the case for other semantics of recursive aggregates).

**Proposition 6 (Stability of Arithmetics).** *Let $f$ be an aggregate name, $S$ a set expression, $y$ an integer and $\odot$ an arithmetic relation.For any program $P_1$, the program $P_2$ obtained from $P_1$ by replacing a rule*

$$head \leftarrow body, f(S) \odot y$$

*by*

$$head \leftarrow body, f(S) = Z, Z \odot y.$$

*is strongly equivalent to $P_1$.*

Two programs $\Pi_1$ and $\Pi_2$ are *strongly equivalent* if for any program $\Pi$, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets [44].

### 6.5. Complexity

A comprehensive analysis of the complexity of aggregate programs, i.e., programs of $\mathscr{A}log$ without non-aggregate set atoms, is given in [45]. Here we examine if the addition of non-aggregate set atoms will increase the complexity. Given the inherent complexity caused by disjunctions, we will consider full $\mathscr{A}log$ programs and programs without disjunctions. In both cases, the addition of non-aggregate set atoms does not increase the complexity of the program.

**Proposition 7 (Complexity of $\mathscr{A}log$ Programs).** *The problem of checking if a ground atom a belongs to all answer sets of an $\mathscr{A}log$ program is $\Pi_2^P$ complete.*

**Proposition 8 (Complexity of $\mathscr{A}log$ Programs without Disjunctions).** *The problem of checking if a ground atom a belongs to all answer sets of an $\mathscr{A}log$ program without disjunctions is coNP complete.*

We only consider the cautious reasoning here. Similar techniques combined with the results from [45] can probably be used to prove complexity results for consistency checking, but we do not do it in this paper.

## 7. An Application of Additive Reduct

The previous sections are devoted to the main subject of this paper – the design and investigation of $\mathscr{A}log$. Here we concentrate on one particular step in this development – the introduction of additive reducts. We believe that such a diversion is justified since additive reducts find use beyond the definition of semantics of $\mathscr{A}log$ and thus may deserve a special attention. This section contains an example of one such use. We introduce additive reducts, similar to one of $\mathscr{A}log$, to

- Give a new definition of another important extension of ASP by aggregates called $\mathscr{S}log$ [26].[9]

- Give an $\mathscr{S}log$ like semantics to an extension of the original $\mathscr{S}log$ by allowing disjunctions in the heads of rules and non-aggregate set atoms in their bodies.

We note that [46] extends $\mathscr{S}log$ to non-disjunctive logic programs with abstract constraint atoms which generalize aggregate atoms [47, 48]. The work in [46] is later extended for disjunctive programs in [49]. An alternative definition of semantics for disjunctive programs was mentioned in [50].[10]

The additive reduct based definition for the core $\mathscr{S}log$ is comparatively simple and makes the essential idea underlying $\mathscr{S}log$ stand out. It also allows natural extension of the semantics of more general languages. Further investigation is needed to see the relation of our work here and that in [49, 50]. It is straightforward to extend our definition of semantics for core $\mathscr{S}log$ to that for disjunctive programs with constraint atoms. We conjecture that the extension coincides with the semantics in [49, 50].

### 7.1. Syntax and Semantic of $\mathscr{S}log$

Syntactically, a *core $\mathscr{S}log$ program* can be viewed as an $\mathscr{A}log$ program without disjunctions, classical negations, rules with infinite head or body, partial aggregates and subset relations.

[9] For simplicity, we focus on the core part of $\mathscr{S}log$ syntax, i.e., the $\mathscr{S}log$ without multisets. The whole language can also be covered but since we are not yet sure about importance of including multisets in the language we do not consider them in this paper.

[10] This definition is also reduct based, but it reduces a constraint atom to another constraint atom. Instead, we reduce a constraint atom to regular atoms and thus can "reuse" the definition of ASP with regular atoms.

We next review the notions in $\mathscr{S}log$ semantics. Consider a set $S$ of ground regular atoms and an aggregate atom $agg$. By *a ground aggregate atom*, we mean an aggregate atom containing no free occurrences of any variables. A *ground atom occurs* in $agg$ if it is the instance of some regular atom occurring in $agg$. $Base(agg)$ denotes the set of the ground atoms occurring in $agg$. We define $ta(agg,S) = \{l : l \in S, l$ occurs in $agg\}$, i.e., $S \cap Base(agg)$, and $fa(agg,S) = Base(agg) \setminus S$.

An *aggregate solution* of a ground aggregate atom $agg$ is a pair $\langle S_1, S_2 \rangle$ of disjoint subsets of $Base(agg)$ such that for every set $S$ of regular ground atoms, if $S_1 \subseteq S$ and $S \cap S_2 = \{\}$ then $S \models agg$ (i.e., $agg$ is true in $S$).

Given a core $\mathscr{S}log$ program $P$ and a set $S$ of ground regular atoms, the $\mathscr{S}log$ *reduct* of $P$ with respect to $S$, denoted by ${}^{S}P$, is defined as

$${}^{S}P = \{\, head(r) \leftarrow pos(r), aggs(r) : r \in ground(P), S \cap \{l : not\ l \in neg(r)\} = \{\}\}.$$

where $head(r)$ is the head of rule $r$, $pos(r)$ the set of regular atoms in the body of $r$ that are not preceded by *not* or inside an aggregate atom, $aggs(r)$ the set of aggregate atoms in the body of $r$, and $neg(r) = \{not\ l : not\ l$ occurs in the body of $r\}$.

The *conditional satisfaction* of an atom $a$ with respect to two sets, $I$ and $S$, of regular atoms, denoted by $(I,S) \models a$, is defined as

1. If $a$ is a regular atom, $(I,S) \models a$ if $I \models a$, and
2. If $a$ is an aggregate atom, $(I,S) \models a$ if $\langle I \cap S \cap Base(a), Base(a) \setminus S \rangle$ is an aggregate solution of $a$.

Given a set $A$ of ground atoms (regular or aggregate), $(I,S) \models A$ denotes that for every atom $a \in A$, $(I,S) \models a$.

Given a core $\mathscr{S}log$ program $P$ and a set $S$ of ground regular atoms, for any collection $I$ of ground regular atoms of $P$, the *consequence operator* on $P$ and $S$, denoted by $K_S^P$, is defined as $K_S^P(I) = \{head(r) : r \in {}^{S}P$ and $(I,S) \models body(r)\}$.

A set $S$ of ground regular atoms is an $\mathscr{S}log$ *answer set* of a core $\mathscr{S}log$ program $P$ if $S = lfp(K_S^P)$.

### 7.2. Additive Reduct Based Definition of Core $\mathscr{S}log$ and Its Extension

To illustrate the idea behind the new additive reduct, we consider program $P_2$ in Example 2

```
p(1) :- card{X: p(X)} >= 0.
```

and a set $S = \{p(1)\}$.

To adopt the notion of $\mathscr{A}log$ reduct to $\mathscr{S}log$ we do the following. Instead of replacing $card\{X : p(X)\} >= 0$, which is true in $S$, by $\{p(1)\}$, as in $\mathscr{A}log$, we replace this aggregate atom by its "minimal guarantee support" – a subset $M$ of $S$, which "guarantees" that the set atom is true in any possible expansion of $M$ with atoms of $S$. In our case, $\{\}$ is such a minimal guarantee support of $card\{X : p(X)\} >= 0$. As a result, the new reduct is

```
p(1).
```

Clearly, $S$ is its answer set and thus is an $\mathscr{S}log$ answer set of $P_2$.

The concept of minimal guarantee support is defined as follows.

**Definition 15 (Minimal Guarantee Support).** Let $S$ be a set of ground regular atoms of $\Pi$, and *agg* be an aggregate atom. $M$ is a *minimal guarantee support* for *agg* in $S$ if

- $M \subseteq S$,
- every $S_1$, such that $M \subseteq S_1 \subseteq S$, satisfies *agg*, and
- no $M_1$, such that $M_1 \subset M$, satisfies the first two conditions. □

Now we introduce the reduct based on minimal guarantee support, called *S-reduct*.

**Definition 16 (S-reduct, S-answer Sets).** An *S-reduct* of an aggregate program $\Pi$ with respect to a set $A$ of ground regular literals is obtained from $\Pi$ by

1. removing rules containing set atoms which are *false* or *undefined* in $A$, and
2. replacing every remaining set atom $C$ in the body of the rule by a minimal guarantee support for $C$ in $A$.

$A$ is an *S-answer set* of $\Pi$ if $A$ is an answer set of an S-reduct of $\Pi$ with respect to $A$. □

**Example 28.** Consider now a program $P_{13}$

```
p(1).
p(3) :- card{X : p(X)} >= 2.
p(2) :- card{X : p(X)} >= 2.
```

It has two possible answer sets: $A_1 = \{p(1)\}$ and $A_2 = \{p(1), p(2), p(3)\}$. In $A_1$, no set atoms occurring in the program are true, and thus, the S-reduct of the program with respect to $A_1$ is $p(1)$. Consequently, $A_1$ is an S-answer set of $P_{13}$. In $A_2$, there are three minimal guarantee supports for the set atom occurring in $P_{13}$: $M_1 = \{p(1), p(2)\}$, $M_2 = \{p(1), p(3)\}$, and $M_3 = \{p(2), p(3)\}$. Hence, the program has nine S-reducts of $P_{13}$ with respect to $A_2$. Each reduct is of the form

`p(1).`     `p(3) :-` $M_i$ `.`     `p(2) :-` $M_j$ `.`

where $i, j \in 1..3$. Clearly, the last two rules are useless and hence $A_2$ is not an answer set of this reduct. Consequently $A_2$ is not an S-answer set of $P_{13}$. □

The following result shows the equivalence between the S-answer sets and $\mathcal{S}log$ semantics on core $\mathcal{S}log$ programs.

**Proposition 9.** *Let* $\Pi$ *be a core* $\mathcal{S}log$ *program. A set is an* $\mathcal{S}log$ *answer set of* $\Pi$ *iff it is an S-answer set of* $\Pi$.

The reduct based approach to defining $\mathcal{S}log$ like semantics seems to be simple: the S-reduct and S-answer set definitions are very close to the classical definitions of reducts and answer sets, and the essential idea underlying $\mathcal{S}log$ is captured by the intuitive and simple concept of minimal guaranteed support.

The extension of the core $\mathcal{S}log$ semantics to programs with disjunction does not require any changes in the definitions above. However, it is not immediately clear to us how the original definitions of $\mathcal{S}log$ semantics can be extended in a straightforward manner to cover disjunction. To extend the semantics to programs with other set atoms in the bodies we only need to replace the aggregate atom in the definition of minimal guaranteed support by arbitrary set atom.

It is worth to note that our main purpose of this section is to demonstrate the capacity of our new reduct technique in defining other semantics. As for the semantics of programs with set atoms, we prefer $\mathcal{A}log$ one.

## 8. Related Work

There are multiple approaches to introducing aggregates in logic programming languages under the answer sets semantics [28, 4, 48, 51, 27, 52, 53, 54, 25, 55, 34, 26, 56, 57, 49, 58, 50, 59, 60, 61]. Two of these semantics [54, 27], which agree for programs without negated aggregates [62], are implemented in popular ASP solvers [12] and [9]. In [61], it was shown that both semantics were equivalent for a large class of programs which includes non-recursive aggregates, even for programs with negative aggregates. To ensure compatibility of various solvers the ASP-Core document [63], produced in 2012 – 2015 by the ASP Standardization Working Group and intended as a specification for the behavior of answer set programming systems, only allows non-recursive use of aggregates.

All these important works helped to discover subtle and fundamental difficulties related to the notion of aggregates and, of course, had an important impact on the design of our language. In addition, our paper was significantly influenced by the original work by Poincaré, Russell, Feferman [64], and others on VCP in set theory. Substantial role was also played by the principles of language design advocated by Dijkstra, Hoare, Wirth, McCarthy and others (see for instance, [65]).

Switching from the standard ASP rules to infinitary ones was adopted from [66], where the authors explained the semantics of some constructs of Gringo (the input language of many ASP systems including CLINGO [67]) in terms of infinitary formulas of Truszczynski [68].
The crucial notion of aggregate reduct of $\mathscr{A}log$ was influenced by the first additive reduct introduced for defining the semantics of Epistemic Specification in [69].

Since the first introduction of the original $\mathscr{A}log$ in [30],[11] there has been a substantive amount of work investigating the language. We briefly describe some of this work.

In [45, 70], the authors study the complexity of both coherence testing and cautious reasoning in original $\mathscr{A}log$. They also propose methods to compile such programs into $\mathscr{F}log$ programs and develop a prototype implementation of the original $\mathscr{A}log$ based on their compilation methods. [41] helps us to realize the close relationship between $\mathscr{A}log$ and $\mathscr{S}log$ and argumentation theory [71] which provides us with additional knowledge about aggregate programs. Together with completion of a logic program, loop formulas for the program provide not only an alternative characterization of the answer sets of the program but also an approach of computing answer sets using efficient satisfiability solvers. Loop formulas are defined for $\mathscr{A}log$ programs in [72]. A characterization of various answer set based semantics for logic programs with aggregates is proposed in [73]. Under this characterization, the connection among $\mathscr{A}log$, $\mathscr{F}log$ and $\mathscr{S}log$ becomes straightforward. Answer set semantics has been extended to programs with syntax similar to propositional formulas [74]. In [54], propositional formulas are extended with aggregates and its answer set based semantics is connected to the logic of here-

[11] Recall that this paper contained no set constructs except that of aggregates.

and-there, which facilitates the study of properties of logic programs. [42] expands our treatment of aggregates to propositional formulas with aggregates and compares the resulting semantics with existing work [54]. A new extension of functional ASP is proposed to allow evaluable functions, arbitrary formulas, and the use of set expressions as arguments for any predicate or function [75]. It shows that when restricted to $\mathscr{A}log$ syntax, its semantics coincides with $\mathscr{A}log$.

## 9. Conclusion

In this paper we

1. Describe an extension $\mathscr{A}log$ of the original Answer Set Prolog [3] by:
    - Means of forming (possibly infinite) sets based on a particularly simple and restrictive formalization of the Vicious Circle Principle.
    - Aggregates, understood as functions on sets.
    - Subset relation which, when used in the bodies of rules, concisely express a specific form of universal quantification. When used in the heads the construct formalizes standard mathematical expressions such as "let $p$ be an arbitrary subset of set $q$".
    - Rules with infinite heads and bodies.
2. Give examples of the use of $\mathscr{A}log$ for knowledge representation and prove a number of important properties of its programs.
3. List some general principles of language design and illustrate the important roles these principles played in the design of $\mathscr{A}log$.
4. Introduce a notion of additive reduct (which generalizes the $\mathscr{A}log$ reduct – the main technical tool used to define the semantics of $\mathscr{A}log$) and show how an additive reduct can be used to define other semantics for aggregates such as $\mathscr{S}log$ in an intuitive, simple and elegant manner.

Even though we want $\mathscr{A}log$ to be a language suitable for serious applications, our main emphasis is on theoretical investigation of use and formation of sets in logic programing, on the principles of language design and their applications, and on the creation of a good language for teaching declarative programming and relevant aspects of knowledge representation. This puts substantial premium on clarity and simplicity of the language constructs. Consequently, we attempt to explain why existing extensions of ASP with features similar to that of $\mathscr{A}log$

do not always satisfy this criteria. In particular, we point out that the lack of stability of $\mathscr{S}log$ and $\mathscr{F}log$, or insufficiently declarative reading of choice rules of [67] prevents us from advocating these languages for use in teaching. There, of course, remains an important unanswered question: is expressive power of $\mathscr{A}log$ sufficient or new extensions of $\mathscr{A}log$ by set constructs will be needed to make it a standard ASP extension by sets. We did not discuss, for instance, inclusion in the language of set operations and rules with variables ranging over sets (in the style of [76]), etc. Partly it is due to natural space limitations. But partly it is because we do not want to introduce any new constructs without convincing examples of their use. So far, we are on the fence on this one. We hope that with more experience we will know if such extensions are justified. Of course, to make $\mathscr{A}log$ a language of choice for practical applications there should be a very efficient $\mathscr{A}log$ solver. One possible way to do that is to consider the algorithms for computing answer sets of ASP programs with $\mathscr{A}log$ aggregates which have already been introduced in [30] and [70] and investigate if they can be extended and efficienty implemented for the richer version of $\mathscr{A}log$ presented in this paper. Development of these and other possible approaches to the design and implementation of $\mathscr{A}log$ solvers is an important subject for future work.

## 10. Acknowledgment

We would like to thank Evgenii Balai, Amelia Harrison, Patrick Kahl, Vladimir Lifschitz, Tran Cao Son, Mirek Truszczynski and the anonymous reviewers for useful comments which helped to improve the quality of the paper. The authors' work was partially supported by NSF grant IIS-1018031 and CNS-1359359.

## References

[1] M. Gelfond, On stratified autoepistemic theories, in: Proceedings of Sixth National Conference on Artificial Intelligence, 1987, pp. 207–212.

[2] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Proceedings of ICLP-88, 1988, pp. 1070–1080.

[3] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Computing 9 (3/4) (1991) 365–386.

[4] I. Niemela, P. Simons, T. Soininen, Extending and implementing the stable model semantics, Artificial Intelligence 138 (1–2) (2002) 181–234.

[5] F. Lin, Y. Zhao, ASSAT: Computing answer sets of a logic program by SAT solvers, Artificial Intelligence 157(1-2) (2004) 115–137.

[6] Y. Lierler, cmodels - sat-based disjunctive answer set solver, in: LPNMR, 2005, pp. 447–451.

[7] Z. Lin, Y. Zhang, H. Hernandez, Fast SAT-based answer set solver., in: Proceedings of AAAI, 2006, pp. 92–97.

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

[9] W. Faber, G. Pfeifer, N. Leone, T. Dell'Armi, G. Ielpa, Design and implementation of aggregate functions in the dlv system, Theory and Practice of Logic Programming 8 (5-6) (2008) 545–580.

[10] M. Gebser, B. Kaufmann, A. Neumann, T. Schaub, Conflict-driven answer set enumeration, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2007, pp. 136–148.

[11] T. Janhunen, I. Niemelä, Compact translations of non-disjunctive answer set programs to propositional clauses, in: Logic programming, knowledge representation, and nonmonotonic reasoning, Springer, 2011, pp. 111–130.

[12] M. Gebser, B. Kaufmann, T. Schaub, Conflict-driven answer set solving: From theory to practice, Artificial Intelligence 187 (2012) 52–89.

[13] M. Alviano, C. Dodaro, W. Faber, N. Leone, F. Ricca, Wasp: A native asp solver based on constraint learning, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2013, pp. 54–66.

[14] M. Maratea, L. Pulina, F. Ricca, Multi-level algorithm selection for asp, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2015, pp. 439–445.

[15] M. Balduccini, C. Baral, Y. Lierler, Knowledge representation and question answering, chapter 21. handbook of knowledge representation (2006).

[16] G. Brewka, T. Eiter, M. Truszczynski, Answer Set Programming at a glance, Commun. ACM 54 (12) (2011) 92–103.

[17] E. Erdem, M. Gelfond, N. Leone, Applications of Answer Set Programming, AI Magazine 37 (3) (2016) 53–68.

[18] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, E. C. Teppan, Industrial applications of answer set programming, KI-Künstliche Intelligenz 32 (2-3) (2018) 165–176.

[19] M. Balduccini, M. Gelfond, Logic programs with consistency-restoring rules, in: International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series, Vol. 102, The AAAI Press, 2003.

[20] C. Baral, M. Gelfond, J. N. Rushton, Probabilistic reasoning with answer sets, Theory and Practice of Logic Programming 9 (1) (2009) 57–144.

[21] D. Pearce, A new logical characterization of stable models and answer sets, in: Non-monotonic Extension of Logic Programming, Vol. 1216 of LNCS, Springer Verlag, 1997, pp. 57–70.

[22] P. Ferraris, J. Lee, V. Lifschitz, A new perspective on stable models, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2007, pp. 372–379.

[23] I. Niemelä, P. Simons, Extending the smodels system with cardinality and weight constraints, in: Logic-based artificial intelligence, Springer, 2000, pp. 491–521.

[24] F. Buccafurri, N. Leone, P. Rullo, Strong and weak constraints in disjunctive datalog, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 1997, pp. 2–17.

[25] N. Pelov, M. Denecker, M. Bruynooghe, Well-fouded and stable semantics of logic programs with aggregates, Theory and Practice of Logic Programming 7 (2007) 355–375.

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

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

[28] M. Gelfond, Representing Knowledge in A-Prolog, in: A. C. Kakas, F. Sadri (Eds.), Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II, Vol. 2408, Springer Verlag, Berlin, 2002, pp. 413–451.

[29] D. B. Kemp, P. J. Stuckey, Semantics of logic programs with aggregates, in: ISLP, Vol. 91, Citeseer, 1991, pp. 387–401.

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

[31] M. Gelfond, Y. Zhang, Vicious circle principle and formation of sets in asp based languages, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2017, pp. 146–159.

[32] I. S. Mumick, H. Pirahesh, R. Ramakrishnan, The magic of duplicates and aggregates, in: Proceedings of the 16th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 1990, pp. 264–277.

[33] K. A. Ross, Y. Sagiv, Monotonic aggregation in deductive database, J. Comput. Syst. Sci. 54 (1) (1997) 79–97.

[34] T. C. Son, E. Pontelli, I. Elkabani, An unfolding-based semantics for logic programming with aggregates, CoRR abs/cs/0605038. `arXiv:cs/0605038`. URL `http://arxiv.org/abs/cs/0605038`

[35] M. Gelfond, H. Przymusinska, On consistency and completeness of autoepistemic theories, Fundamenta Informaticae 16 (1) (1992) 59–92.

[36] V. Lifschitz, H. Turner, Splitting a logic program, in: Proceedings of the 11th International Conference on Logic Programming (ICLP94), 1994, pp. 23–38.

[37] H. Turner, Splitting a Default Theory, in: Proceedings of AAAI-96, 1996, pp. 645–651.

[38] C. Baral, M. Gelfond, Logic Programming and Knowledge Representation, Journal of Logic Programming 19 (20) (1994) 73–148.

[39] M. Gelfond, H. Przymusinska, On consistency and completeness of autoepistemic theories, Fundam. Inf. 16 (1).

[40] K. R. Apt, H. A. Blair, A. Walker, Towards a theory of declarative knowledge, in: Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, 1988, pp. 89–148.

[41] M. Alviano, W. Faber, Stable model semantics of abstract dialectical frameworks revisited: A logic programming perspective, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence. IJCAI Organization, Buenos Aires, Argentina, 2015, pp. 2684–2690.

[42] P. Cabalar, J. Fandinno, T. Schaub, S. Schellhorn, Gelfond-Zhang aggregates as propositional formulas, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2017, pp. 117–131.

[43] M. Gelfond, Y. Zhang, Vicious circle principle and logic programs with aggregates, arXiv preprint arXiv: 1808.07050.

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

[45] M. Alviano, N. Leone, Complexity and compilation of gz-aggregates in answer set programming, Theory and Practice of Logic Programming 15 (4-5) (2015) 574–587.

[46] T. C. Son, E. Pontelli, P. H. Tu, Answer sets for logic programs with arbitrary abstract constraint atoms, J. Artif. Intell. Res. (JAIR) 29 (2007) 353–389.

[47] V. W. Marek, M. Truszczynski, Logic programs with abstract constraint atoms, in: Proceedings of AAAI04, 2004, pp. 86–91.

[48] V. W. Marek, J. B. Remmel, Set constraints in logic programming, in: Logic Programming and Nonmonotonic Reasoning, Springer, 2004, pp. 167–179.

[49] Y. Shen, J. You, L. Yuan, Characterizations of stable model semantics for logic programs with arbitrary constraint atoms, Theory and Practice of Logic Programming 9 (4) (2009) 529–564.

[50] G. Liu, R. Goebel, T. Janhunen, I. Niemelä, J.-H. You, Strong equivalence of logic programs with abstract constraint atoms, in: Logic Programming and Nonmonotonic Reasoning, Springer, 2011, pp. 161–173.

[51] W. Faber, N. Leone, G. Pfeifer, Recursive aggregates in disjunctive logic programs: Semantics and complexity, in: Proceedings of the 8th European Conference on Artificial Intelligence (JELIA 2004), 2004, pp. 200–212.

[52] P. Ferraris, V. Lifschitz, Weight constraints as nested expressions, Theory and Practice of Logic Programming 5 (1-2) (2005) 45–74.

[53] P. Ferraris, Answer sets for propositional theories, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2005, pp. 119–131.

[54] P. Ferraris, Logic programs with propositional connectives and aggregates, ACM Transactions on Computational Logic (TOCL) 12 (4) (2011) 25.

[55] B. Bogaerts, J. Vennekens, M. Denecker, Grounded fixpoints and their applications in knowledge representation, Artif. Intell. 224 (2015) 51–71. doi:10.1016/j.artint.2015.03.006.
URL https://doi.org/10.1016/j.artint.2015.03.006

[56] J. Lee, V. Lifschitz, R. Palla, A reductive semantics for counting and choice in answer set programming, in: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008, 2008, pp. 472–479.

[57] J. Lee, Y. Meng, On reductive semantics of aggregates in answer set programming, in: Logic Programming and Nonmonotonic Reasoning, Springer, 2009, pp. 182–195.

[58] L. Liu, E. Pontelli, T. C. Son, M. Truszczynski, Logic programs with abstract constraint atoms: The role of computations, Artif. Intell. 174 (3-4) (2010) 295–315.

[59] Y.-D. Shen, K. Wang, T. Eiter, M. Fink, C. Redl, T. Krennwallner, J. Deng, Flp answer set semantics without circular justifications for general logic programs, Artificial Intelligence 213 (2014) 1–41.

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

[61] A. Harrison, V. Lifschitz, Relating two dialects of answer set programming, in: Working Notes of the 17th International Workshop on Non-Monotonic Reasoning, 2018.
URL http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=127711

[62] M. Alviano, W. Faber, M. Gebser, Rewriting recursive aggregates in answer set programming: back to monotonicity, Theory and Practice of Logic Programming 15 (4-5) (2015) 559–573.

[63] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, T. Schaub, Asp-core-2: Input language format (2015).

[64] S. Feferman, Predicativity, http://math.stanford.edu/~feferman/papers/ (2002).

[65] N. Wirth, On the design of programming languages., in: IFIP Congress, Vol. 74, 1974, pp. 386–393.

[66] A. J. Harrison, V. Lifschitz, F. Yang, The semantics of gringo and infinitary propositional formulas, in: KR, 2014.

[67] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, S. Thiele, P. Wanko, Potassco user guide, Institute for Informatics, University of Potsdam, Version 2.2.0 (2019).

[68] M. Truszczynski, Connecting first-order asp and the logic fo (id) through reducts, in: Correct Reasoning, Springer, 2012, pp. 543–559.

[69] M. Gelfond, New semantics for epistemic specifications, in: Logic Programming and Nonmonotonic Reasoning, Springer, 2011, pp. 260–265.

[70] M. Alviano, N. Leone, On the properties of gz-aggregates in answer set programming., in: IJCAI, 2016, pp. 4105–4109.

[71] H. Strass, Approximating operators and semantics for abstract dialectical frameworks, Artificial Intelligence 205 (2013) 39–70.

[72] C. Li, Y. Wang, R. Feng, Q. Li, Loop formulas for alog answer set programs, in: Proceedings of the 2017 International Conference on Computer Science and Artificial Intelligence, ACM, 2017, pp. 38–42.

[73] Y. Zhang, M. Rayatidamavandi, A characterization of the semantics of logic programs with aggregates., in: IJCAI, 2016, pp. 1338–1344.

[74] V. Lifschitz, L. R. Tang, H. Turner, Nested expressions in logic programs, Annals of Mathematics and Artificial Intelligence 25 (3-4) (1999) 369–389.

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

[76] A. Dovier, E. Pontelli, G. Rossi, Intensional sets in CLP, in: Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings, 2003, pp. 284–299.

[77] S. T. Erdogan, V. Lifschitz, Definitions in answer set programming, in: V. Lifschitz, I. Niemel"a (Eds.), Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), 2004, pp. 114–126.

[78] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.

[79] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Annals of Mathematics and Artificial Intelligence 15 (3-4) (1995) 289–323.

## Appendix

In this appendix, given an $\mathscr{A}log$ program $\Pi$, a set $A$ of literals and a rule $r \in \Pi$, we use $R_{\mathscr{X}}(r,A)$, where $\mathscr{X}$ is either $\mathscr{A}$ or $\mathscr{S}$, to denote the set of rule(s) obtained from $r$ in the $\mathscr{X}$-reduct (i.e., aggregate reduct or set introduction reduct) of $\Pi$ with respect to $A$. $R_{\mathscr{X}}(r,A)$ is empty if $r$ is discarded in the $\mathscr{X}$-reduct. $R_{\mathscr{X}}(r,A)$ may consist of one rule (in $\mathscr{A}$-reduct) or more than one rule (in $\mathscr{S}$-reduct). We use $R_{\mathscr{X}}(\Pi,A)$ to denote the $\mathscr{X}$-reduct of $\Pi$, i.e., $\cup_{r\in\Pi} R_{\mathscr{X}}(r,A)$.

To prove Proposition 1 we will need two auxiliary lemmas.

**Lemma 1.** *Let $\Pi$ be a ground $\mathscr{A}log$ program which contains no occurrences of set atoms, $A$ be an answer set of $\Pi$ and $R$ be the set of all rules of $\Pi$ whose bodies are satisfied by $A$. Then $A$ is a minimal (with respect to set-inclusion) set of literals which satisfies $R$.*

Proof.

The fact that $A$ satisfies $R$ follows immediately from the definition of answer set. To prove minimality assume that

(1) $B \subseteq A$

(2) $B$ satisfies $R$

and show that

(3) $B$ satisfies the reduct $\Pi^A$.

Consider a rule $head \leftarrow body^A$ from $\Pi^A$ such that

(4) $B$ satisfies $body^A$.

Since $body^A$ contains no default negation, (1) and (4) imply that

(5) $A$ satisfies $body^A$.

By definition of a reduct,

(6) $A$ satisfies $body$

and hence the rule

(7) $head \leftarrow body$ is in $R$.

By (2) and (7), $head$ is satisfied by $B$.

Therefore, $B$ satisfies $head \leftarrow body^A$ and, hence, (3) holds.

Since $A$ is an answer set of $\Pi^A$, (3) implies that

(8) $B = A$

which concludes the proof. □

**Definition 17 (Supportedness).** *Let $A$ be an answer set of a ground program $\Pi$ of $\mathscr{A}log$. We say that a literal $p(t) \in A$ is* supported *by a rule $r$ from $\Pi$ if the body of $r$ is satisfied by $A$ and*

- *$p(t)$ is the only atom in the head of $r$ which is true in $A$, or*
- *$r$ is a set introduction rule defining $p$, and its head is true in $A$.*

To show supportedness for $\mathscr{A}log$ with set atoms we need to first prove this property for $\mathscr{A}log$ programs not containing set atoms (similar result for disjunctive programs with finite rules can be found in [38]).

**Lemma 2 (Supportedness for Programs without Set Atoms).** *Let $A$ be an answer set of a ground program $\Pi$ of $\mathscr{A}log$ which contains no occurrences of set atoms. Then*

1. *$A$ satisfies every rule $r$ of $\Pi$.*
2. *If $p(t) \in A$ then there is a rule $r$ from $\Pi$ which supports $p$.*

Proof:

1. The first clause follows immediately from the definition of an answer set of a program without set atoms.
2. Let $p(t) \in A$.

   To prove the existence of a rule of $\Pi$ supporting $p(t)$ we consider the set $R$ of all rules of $\Pi$ whose bodies are satisfied by $A$.

   Suppose $p(t)$ does not belong to the head of any rule from $R$. Then $A \setminus \{p(t)\}$ also satisfies rules of $R$. (Indeed, suppose that $A \setminus \{p(t)\}$ satisfies the body of a rule from $R$. Then, by definition of $R$, the rule's head is satisfied by $A$. Since the head does not contain $p(t)$, it is also satisfied by $A - \{p(t)\}$.) This contradicts the minimality condition from Lemma 1.)

   Suppose now that for every rule $r \in R$ which contains an occurrence of $p(t)$ in the head $head(r) \cap A \neq \{p(t)\}$. But then $A - \{p(t)\}$ would again satisfy $R$ which would contradict Lemma 1. This concludes the proof of the Lemma 2. □

Now we are ready to prove Proposition 1

**Proposition 1 (Rule Satisfaction and Supportedness).** *Let $A$ be an answer set of a ground $\mathscr{A}log$ program $\Pi$. Then*

1. *$A$ satisfies every rule $r$ of $\Pi$.*
2. *If $p \in A$ then there is a rule $r$ supporting $p$.*

Proof: Let

(1) $A$ be an answer set of $\Pi$, and

$\Pi'$ be the result of the aggregate reduct of the set introduction reduct of $\Pi$ with respect to $A$, i.e., $\Pi' = R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi, A), A)$.

We first prove that $A$ satisfies every rule $r$ of $\Pi$. Let $r$ be a rule of $\Pi$ such that

(2) $A$ satisfies the body of $r$.

Statement (2) implies that every set atom, if there is any, of the body of $r$ is satisfied by $A$. By the definition of the aggregate reduct and set introduction reduct, there must be a non-empty rule $r' \in \Pi'$ such that

(3) $r' \in R_{\mathscr{A}}(R_{\mathscr{S}}(r, A), A)$.

By the definition of aggregate reduct, $A$ satisfies the body of $r$ iff it satisfies that of $r'$. Therefore, (2) and (3) imply that

(4) $A$ satisfies the body of $r'$.

By the definition of answer set of $\mathscr{A}log$, (1) implies that

(5) $A$ is an answer set of $R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi, A), A)$.

Since $R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi, A), A)$ is an ASP program, (3) and (5) imply that

(6) $A$ satisfies $r'$.

Consider two cases on whether $r$ is a set introduction rule.

Case 1: $r$ is not a set introduction rule. Then, $r$ and and $r'$ have the same head. Statements (4) and (6) imply $A$ satisfies the head of $r'$ and thus the head of $r$.

Case 2: $r$ is a set introduction rule. Statement (4) and (6) imply that the head of $r'$ is not empty. Hence, by definition of set introduction reduct, the head of $r$ is satisfied by $A$.

Therefore $r$ is satisfied by $A$, which concludes our proof of the first part of the proposition.

We next prove the second part of the proposition. Consider $p(t) \in A$. (1) implies that $A$ is an answer set of $R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi, A), A)$, i.e., $\Pi'$. By Lemma 2 there is a rule $r' \in \Pi'$ such that

(7) $r'$ supports $p(t)$.

Let $r \in \Pi$ be a rule such that $r' \in R_{\mathscr{A}}(R_{\mathscr{S}}(r, A), A)$. By the definition of aggregate reduct and set introduction reduct,

(8) $A$ satisfies the body of $r$ iff $A$ satisfies that of $r'$.

Consider two cases on whether $r$ is a set introduction rule.

Case 1: $r$ is not a set introduction rule. Then $r$ and $r'$ have the same head. So, (7) and (8) imply that rule $r$ of $\Pi$ supports $p(t)$ in $A$.

Case 2: $r$ is a set introduction rule. Since $r'$ is the result of set introduction reduct (and then aggregate reduct) of $r$, the head of $r$ is of the form $p \odot \{\bar{X} : q(\bar{X})\}$ and is true in $A$. Hence, (8) implies that $r$ supports $p(t)$ in $A$.

Therefore, the second part of the proposition holds. □

**Proposition 2 (Anti-chain Property).** *If* $\Pi$ *is a program without set introduction rules then there are no* $\mathscr{A}$*log answer sets* $A_1$, $A_2$ *of* $\Pi$ *such that* $A_1 \subset A_2$.

Proof: Let us assume that there are $A_1$ and $A_2$ such that

(1) $A_1 \subseteq A_2$. and

(2) $A_1$ and $A_2$ are answer sets of $\Pi$.

We will show that $A_1 = A_2$.

Let $R_1$ and $R_2$ be the aggregate reducts of $\Pi$ with respect to $A_1$ and $A_2$ respectively. Let us first show that $A_1$ satisfies the rules of $R_2$. Consider

(3) $r_2 \in R_2$.

By the definition of aggregate reduct there is $r \in \Pi$ such that

(4) $r_2 = R_{\mathscr{A}}(r, A_2)$.

Consider

(5) $r_1 = R_{\mathscr{A}}(r, A_1)$.

If $r$ contains no aggregate atoms then

(6) $r_1 = r_2$.

By (5) and (6), $r_2 \in R_1$ and hence, by (2) $A_1$ satisfies $r_2$.

Assume now that $r$ contains one set atom $SA$ and is of the form

(7) $h \leftarrow B, SA$

Then $r_2$ has the form

(8) $h \leftarrow B, P_2$

where

(9) $P_2 = \{cond(t) : cond(t) \text{ occurs in } SA, cond(t) \in A_2\}$.

Let

(10) $P_1 = \{cond(t) : cond(t) \text{ occurs in } SA, cond(t) \in A_1\}$.

Consider two cases (11a) and (11b) below:

(11a) $R_{\mathscr{A}}(r, A_1) = \{\}$.

In this case *SA* is not true in *A*. Hence, $P_1 \neq P_2$. Since $A_1 \subseteq A_2$, we have that $P_1 \subset P_2$. Hence, the body of rule (8) is not satisfied by $A_1$. As a result, rule (8) is satisfied by $A_1$.

(11b) $R_{\mathscr{A}}(r, A_1) \neq \{\}$.

Then *SA* is true in $A_1$, and $r_1$ has the form

(12) $h \leftarrow B, P_1$.

Assume that $A_1$ satisfies the body, i.e., $B$ and $P_2$, of rule (8). Then

(14) $P_2 \subseteq A_1$

This, together with (9) and (10) implies

(15) $P_2 \subseteq P_1$.

From (1), (9), and (10) we have $P_1 \subseteq P_2$. Hence

(16) $P_1 = P_2$.

This means that $A_1$ satisfies the body of $r_1$ and hence it satisfies $h$ and, therefore, $r_2$.

Similar argument works for rules containing multiple set atoms and, therefore, $A_1$ satisfies $R_2$.

Since $A_2$ is a minimal set satisfying $R_2$ and $A_1$ satisfies $R_2$ and $A_1 \subseteq A_2$ we have that $A_1 = A_2$.

This completes our proof. □

We need the following lemma to prove the splitting set theorem.

**Lemma 3 (Constraints without Set Atoms).** *Let* $\Pi_1$ *be a ground program and* $\Pi_2$ *be a set of constraints of* $\mathscr{A}log$. *They contain no occurrences of set atoms. A is an answer set of* $\Pi_1 \cup \Pi_2$ *iff A is an answer set of* $\Pi_1$ *and A satisfies* $\Pi_2$.

**Lemma 4 (Constraints).** *Let* $\Pi_1$ *be a ground program of and* $\Pi_2$ *be a set of constraints of* $\mathscr{A}log$. $\Pi_1$ *contains no set introduction rules. A is an answer set of* $\Pi_1 \cup \Pi_2$ *iff A is an answer set of* $\Pi_1$ *and A satisfies* $\Pi_2$.

Proof.

(1) $A$ is an answer set of $\Pi_1 \cup \Pi_2$ iff

(2) $A$ is answer set of $R_{\mathscr{A}}(\Pi_1 \cup \Pi_2, A)$.

By aggregate reduct, we have

(3) $R_{\mathscr{A}}(\Pi \cup \Pi_2, A) = R_{\mathscr{A}}(\Pi_1) \cup R_{\mathscr{A}}(\Pi_2)$.

By (2) and (3), we have

Statement (2) holds iff

(4) $A$ is answer set of $R_{\mathscr{A}}(\Pi_1, A) \cup R_{\mathscr{A}}(\Pi_2, A)$.

By Lemma 3, (4) and that $R_{\mathscr{A}}(\Pi_2, A)$ is a set of constraints, we have

Statement (4) holds iff

(5) $A$ is an answer set of $R_{\mathscr{A}}(\Pi_1, A)$ and $A$ satisfies $R_{\mathscr{A}}(\Pi_2, A)$.

By definition of answer sets,

Statement (5) holds iff

(6) $A$ is an answer of $\Pi_1$ and $A$ satisfies $R_{\mathscr{A}}(\Pi_2, A)$.

By aggregate reduct definition and that $\Pi_2$ are constraints, $A$ satisfies $\Pi_2$ iff $A$ satisfies $R_{\mathscr{A}}(\Pi_2, A)$. Hence,

Statement (6) holds iff

(7) $A$ is an answer of $\Pi_1$ and $A$ satisfies $\Pi_2$.

Hence, we complete the proof. □

**Proposition 3 (Splitting Set Theorem).** *Let $\Pi$ be a ground program, $S$ be a splitting set of $\Pi$, and $\Pi_1$ and $\Pi_2$ be the bottom and the top of $\Pi$ relative to $S$ respectively. Then a set $A$ is an answer set of $\Pi$ iff $A \cap S$ is an answer set of $\Pi_1$ and $A$ is an answer set of $(A \cap S) \cup \Pi_2$.*

Proof.

First consider the case when $\Pi_1$ and $\Pi_2$ contain no set atoms. It is easy to check that, in this case, we can use the proof from [77], since the infinite number of literals in the rules does not affect the arguments used in this proof.

We now consider programs without set introduction rules. By the definition of answer sets and the aggregate reduct

(1) $A$ is an answer set of $\Pi_1 \cup \Pi_2$ iff

(2) $A$ is an answer set of $R_{\mathscr{A}}(\Pi_1, A) \cup R_{\mathscr{A}}(\Pi_2, A)$.

Without loss of generality, we assume the program does not contains set atoms of the form $p \otimes S$ or $S \otimes p$ because they will be replaced by $\{\bar{X} : p(\bar{X})\} \otimes S$ and $S \otimes \{\bar{X} : p(\bar{X})\}$ respectively in the aggregate reduct.

Consider any rule $r$ of $R_{\mathscr{A}}(\Pi_1, A)$. By definition of aggregate reduct, $r$ is the result of replacing every set atom $SA$ of a rule $r'$ of $\Pi$ by the union of $cond(\bar{t})$ such that $cond(\bar{X})$ for some $\bar{X}$ occurs in $SA$ and $cond(\bar{t}) \subseteq A$.

Since $S$ is a splitting set, by definition, the value of every set expression occurring in $r'$ is determined by $S$. Hence, either some user-defined literal in $cond(\bar{t})$ has no potential support in $\Pi$ or every literal of $cond(\bar{t})$ is from $S$. In the former case, rule $r$ is useless because no answer set of $\Pi$ will satisfy $cond(\bar{t})$ and thus the body of $r$, i.e., $R_{\mathscr{A}}(\Pi_1, A)$ is strongly equivalent to $R_{\mathscr{A}}(\Pi_1, A) \setminus \{r\}$.

Let $P'$ be the result of removing all rules that are useless. $R_{\mathscr{A}}(\Pi_1, A) \cup R_{\mathscr{A}}(\Pi_2, A)$ is strongly equivalent to $P' \cup R_{\mathscr{A}}(\Pi_2, A)$. We know all rules of $P'$ are formed using literals of $S$ and no rules of $\Pi_2$ is a potential support of any literal of $S$. Hence $S$ is a splitting set for $P' \cup R_{\mathscr{A}}(\Pi_2, A)$. Since $P'$ and $R_{\mathscr{A}}(\Pi_2, A)$ contain no set atoms, the Splitting Set Theorem on programs without set atoms, implies that (2) holds iff

(3a) $A \cap S$ is an answer set of $P'$ and thus $R_{\mathscr{A}}(\Pi_1, A)$

and

(3b) $A$ is an answer set of $(A \cap S) \cup R_{\mathscr{A}}(\Pi_2, A)$.

We now assume $A$ is an answer set of $\Pi$, (3a) and (3b). We will show that $A \cap S$ is an answer set of $\Pi_1$ and $A$ is an answer set of $(A \cap S) \cup \Pi_2$.

To show the former, we will show $A \cap S$ is an answer set of $R_{\mathscr{A}}(\Pi_1, A \cap S)$.

For any set expression $\{X : cond(\bar{X})\}$ occurring in any rule of $\Pi$, let $E_1 = \{t : cond(t) \subseteq A\}$ and $E_2 = \{t : cond(t) \subseteq A \cap S\}$. We show,

(4) $E_1 = E_2$

by contradiction. Assume $E_1 \neq E_2$. The only case is that there is $t_1$ such that $cond(t_1) \subseteq A$ but $cond(t_1) \nsubseteq A \cap S$. Since the set expression is determined by $S$, there must be some literal $l$ of $cond(t_1)$ that has no potential support in $\Pi$. By Proposition 1, $l \notin A$, contradicting $l \in A$. As a result of (4), $SA$ is true in $A$ iff $SA$ is true in $A \cap S$.

Similar to the proof for (4), we can show

(5) For any $cond(\bar{X})$ occurring in $\Pi$, $\{cond(t) : cond(t) \subseteq A\} = \{cond(t) : cond(t) \subseteq A \cap S\}$.

By (4) and (5), we have $R_{\mathscr{A}}(\Pi_1, A) = R_{\mathscr{A}}(\Pi_1, A \cap S)$. By (3a) and the definition of answer sets,

(6) $A \cap S$ is an answer set of $\Pi_1$.

By aggregate reduct definition, $R_{\mathscr{A}}((A \cap S) \cup \Pi_2, A) = (A \cap S) \cup R_{\mathscr{A}}(\Pi_2, A)$. Hence, by (3b) and answer set definition,

(7) $A$ is an answer set of $(A \cap S) \cup \Pi_2$.

By (6) and (7), we proved the necessary condition of the theorem for programs without set introduction rules.

We next assume (6) and (7) for any $A$, we will show (3a) and (3b).

For any set expression $\{X : cond(\bar{X})\}$ occurring in any rule of $\Pi$, let $E_1 = \{t : cond(t) \subseteq A\}$ and $E_2 = \{t : cond(t) \subseteq A \cap S\}$. We show,

(8) $E_1 = E_2$

by contradiction. Assume $E_1 \neq E_2$. The only case is that there is $t_1$ such that $cond(t_1) \subseteq A$ but $cond(t_1) \not\subseteq A \cap S$. Since the set expression is determined by $S$, there must be some literal $l$ of $cond(t_1)$ that has no potential support in $\Pi$. Hence, $l \notin A \cap S$ (because of (6)) and thus $l \notin A$ (because of (7)), contradicting $l \in A$. As a result of (8), $SA$ is true in $A$ iff $SA$ is true in $A \cap S$.

Similar to the proof for (8), we can show

(9) For any $cond(\bar{X})$ occurring in $\Pi$, $\{cond(t) : cond(t) \subseteq A\} = \{cond(t) : cond(t) \subseteq A \cap S\}$.

By (8) and (9), we have $R_{\mathscr{A}}(\Pi_1, A) = R_{\mathscr{A}}(\Pi_1, A \cap S)$. Hence, (6) implies

(10a) $A \cap S$ is an answer set of $R_{\mathscr{A}}(\Pi_1, A)$.

By aggregate reduct definition, $R_{\mathscr{A}}((A \cap S) \cup \Pi_2, A) = (A \cap S) \cup R_{\mathscr{A}}(\Pi_2, A)$. Hence, by (7) and answer set definition,

(10b) $A$ is an answer set of $(A \cap S) \cup R_{\mathscr{A}}(\Pi_2, A)$.

By (10a) and (10b) (and (1) iff 5(a) and 5(b)), the sufficient condition of the theorem is proved for the programs without set introduction rules.

Now assume $\Pi$ is an arbitrary program. By definition of answer sets, we have

(11) $A$ is an answer set of $\Pi$ iff

(12) $A$ is an answer set of $R_{\mathscr{S}}(\Pi_1 \cup \Pi_2, A)$.

By definition of set introduction reduct,

(13) $R_{\mathscr{S}}(\Pi_1 \cup \Pi_2, A) = R_{\mathscr{S}}(\Pi_1, A) \cup R_{\mathscr{S}}(\Pi_2, A) = \Pi_1' \cup \Pi_1'' \cup R_{\mathscr{S}}(\Pi_2, A)$.
where $\Pi_1''$ are the constraints of $R_{\mathscr{S}}(\Pi_1, A)$, and $\Pi_1' = R_{\mathscr{S}}(\Pi_1, A) \setminus \Pi_1''$.

We show there is no rule of $R_{\mathscr{S}}(\Pi_2,A)$ whose head contains a literal of $S$. Assume there is such a rule $r$ containing $p(t) \in S$. Let $r_2 \in \Pi_2$ be the rule from which $r$ is obtained. $r_2$ is a potential support of $p(t)$ and thus should belong to $\Pi_1$ and not $\Pi_2$, contradicting $r_2 \in \Pi_2$. One can show that $S$ is a splitting set for $R_{\mathscr{S}}(\Pi_1 \cup \Pi_2,A)$. The bottom of $R_{\mathscr{S}}(\Pi_1 \cup \Pi_2,A)$ relative to $S$ is $\Pi_1'$, and the top is $\Pi_1'' \cup R_{\mathscr{S}}(\Pi_2,A)$. Hence, (12) holds iff

(14) $A \cap S$ is an answer set of $\Pi'$, and

(15) $A$ is an answer set of $(A \cap S) \cup \Pi_1'' \cup R_{\mathscr{S}}(\Pi_2,A)$.

We show $R_{\mathscr{S}}(\Pi_1,A) = R_{\mathscr{S}}(\Pi_1,A \cap S)$ when

(16) $A$ is an answer set of $\Pi$ or $A \cap S$ is an answer set of $\Pi_1$ and $A$ is an answer set of $(A \cap S) \cup \Pi_2$.

For any non-set introduction rule $r \in \Pi_1$, $r \in R_{\mathscr{S}}(\Pi_1,A)$ iff $r \in R_{\mathscr{S}}(\Pi_1,A \cap S)$. For any set introduction rule $r \in \Pi_1$, it is a potential support of some literal of $S$. Let $r$ define $p$. All literals of $p$ are in $S$. Let $p \otimes \{X : q(X)\}$ be the head of $r$.

(17) When $A$ is an answer set of $\Pi$, the head of $r$ is true in $A$ iff it is true in $A \cap S$,

because of the following. Since $S$ is splitting set, for every ground term $t$, $p(t) \in S$, and thus

(18) $\{p(t) : p(t) \in A\} = \{p(t) : p(t) \in A \cap S\}\}$.

Since $S$ is a splitting set, the value of $\{X : q(X)\}$ is determined by $S$ and thus, for any ground term $t$, $q(t) \in S$ or $q(t)$ has no potential support in $\Pi$. Hence, when $q(t) \notin S$, it has no potential support in $\Pi$ and thus is not in any answer set of $\Pi$. Hence,

(19) $\{q(t) : q(t) \in A\} = \{q(t) : q(t) \in A \cap S\}$.

By (18) and (19), we have (17). Similarly, we have

(20) when $A \cap S$ is an answer set of $\Pi_1$ and $A$ is an answer set of $(A \cap S) \cup \Pi_2$, the head of $r$ is true in $A$ iff it is true in $A \cap S$.

By (17) and (20), we have (assuming condition (16))

(21) $R_{\mathscr{S}}(\Pi_1,A) = R_{\mathscr{S}}(\Pi_1,A \cap S)$.

Since $S$ is a splitting set of $\Pi$, every regular literal belonging to rules of $\Pi''$ is in $S$ and all set expressions of $\Pi''$ are determined. Therefore, assuming condition (16), $A$ satisfies $\Pi''$ iff $A \cap S$ satisfies $\Pi''$. By (14), (15) and Lemma4, we have $A \cap S$ is an answer set of $\Pi' \cup \Pi''$, i.e., $R_{\mathscr{S}}(\Pi_1,A)$, and thus by (21),

(22) $A \cap S$ is an answer set of $R_{\mathscr{S}}(\Pi_1,A \cap S)$, i.e., $A \cap S$ is an answer set of $\Pi_1$.

By Lemma4, (15) implies $A$ is an answer set of $(A \cap S) \cup R_{\mathscr{S}}(\Pi_2,A)$, and thus

(23) $A$ is an answer set of $R_{\mathscr{S}}((A \cap S) \cup \Pi_2,A)$, i.e., $A$ is an answer set of $(A \cap S) \cup \Pi_2$.

In a similar manner, we can prove (14) and (15) from (22) and (23). In summary, (11) iff (22) and (23). Hence, we complete the proof. □

**Corollary 1.** *A is an answer set of* $\Pi$ *iff* $A \cap S$ *is an answer set of* $\Pi_1$ *and* $A$ *is an answer set of* $(A \cap S) \cup Red(\Pi_2, A \cap S)$.

Proof. Let us fix $A$ and denote $(A \cap S) \cup \Pi_2$ by $T_1$ and $(A \cap S) \cup Red(\Pi_2, A \cap S)$ by $T_2$. By the splitting set theorem

1. $A$ is an answer set of $\Pi$ iff

2a. $A \cap S$ is an answer set of $\Pi_1$ and

2b. $A$ is an answer set of $T_1(A)$.

By the definition of answer set,

3. (2b) holds iff $A$ is an answer set of $T_1^R(A)$.

Similarly,

4. $A$ is an answer set of $T_2$ iff $A$ is an answer set of $T_2^R(A)$.

Examination of the definitions of $\mathscr{A}log$ reduct and the splitting set reduct *Red* shows the following relationship between $T_1^R$ and $T_2^R$:

5. If $head \leftarrow body$ is in $T_1^R$ then $head \leftarrow (body \setminus S)$ is in $T_2^R$.

6. If $head \leftarrow body$ is in $T_2^R$ then there is a rule $head \leftarrow body_0$ in $T_1^R$ such that $body = body_0 \setminus S$.

So $A$ satisfies rules of $T_1^R$ iff it satisfies rules of $T_1^R$. Moreover, this is true for every set containing $A \cap S$ which implies that

7. $A$ is an answer set of $T_1^R(A)$ iff $A$ is an answer set of $T_2^R(A)$.

This, together with (3) and (4) implies the conclusion of the corollary. □

In what follows we will prove the stratification result, which needs Lemma 5 to 9. It will be useful to extend leveling from atoms to rules and programs. Let $\Pi$ be a program with a stratifying leveling $\| \; \|$. By the definition of stratification, if $l_1$ and $l_2$ are potentially supported by a rule $r \in \Pi$ then $\| l_1 \| = \| l_2 \|$. Moreover, since $\Pi$ has no constraints, every rule $r$ of $\Pi$ has at least one $l$ potentially supported by it. Therefore we can expand leveling $\| \; \|$ to rules of $\Pi$ by making $\| r \| = \| l \|$. The leveling can be further expanded to programs: $\| \Pi \|$ is the smallest ordinal which is greater than or equal to levels of all rules of $\Pi$. For instance, consider a program $\Pi$ consisting of rules

```
p(0).
p(N+1) :- not p(N).
```

where $N$ ranges over natural numbers, and its leveling $\| p(i) \| = i$. Clearly, $\| \Pi \| = \omega$.

We will need the following notation:

**Definition 18.** *Let $\| \ \|$ be a stratification of $\Pi$ with maximal ordinal $\lambda$. For every ordinal $\alpha \leq \lambda$*

$$\Pi_\alpha =_{def} \{r \in \Pi : \| r \| \leq \alpha\}$$

*and*

$$\Phi_\alpha =_{def} \{r \in \Pi : \| r \| = \alpha\}.$$

**Lemma 5 (*$\mathscr{A}$log* Reduct).** *Let $\| \ \|$ be a stratification of $\Pi$ with maximal ordinal $\lambda$. For any $\alpha \leq \lambda$, if*

$$\Pi_\alpha = \bigcup_{\beta<\alpha} \Pi_\beta$$

*then*

$$\Pi^R_\alpha(A) = \bigcup_{\beta<\alpha} \Pi^R_\beta(A)$$

Proof.

Prove $\subseteq$.

For any $r \in \Pi^R_\alpha(A)$, there exists $r' \in \Pi_\alpha$ from which $r$ is obtained. In this case, we say that $r$ is the reduct of $r'$ wrt $A$. Since $\Pi_\alpha = \bigcup_{\beta<\alpha} \Pi_\beta$, there exists $\beta_1 < \alpha$, $r' \in \Pi_{\beta_1}$. Since $r$ is the reduct of $r'$, $r \in \Pi^R_{\beta_1}(A)$. Hence $r \in \bigcup_{\beta<\alpha} \Pi^R_\beta(A)$.

Prove $\supseteq$.

For any $r \in \bigcup_{\beta<\alpha} \Pi^R_\beta(A)$, there exists $\beta_1 < \alpha$ such that $r \in \Pi^R_{\beta_1}(A)$. Hence there exists $r' \in \Pi_{\beta_1}$ such that $r$ is a reduct of $r'$ wrt $A$. Since $\beta_1 < \alpha$, $r' \in \Pi_\alpha$ because $\Pi_\alpha = \bigcup_{\beta<\alpha} \Pi_\beta$. Since $r$ is the reduct of $r'$, t $r \in \Pi^R_\alpha(A)$. □

**Definition 19 (Auxiliary Reduct).** *Auxiliary reduct $AR(\Pi, B)$* of $\Pi$ with respect to set $B$ is obtained as follows:

(a) Remove all rules with set atoms in their bodies which are false or undefined in $B$.

(b) For every remaining set expression, say, $\{X : q(X)\}$ in a rule $r$ add the set $\{q(t) : q(t) \in B\}$ to the body of $r$.

(c) Remove all set atoms from the bodies of the rules.

(d) Replace a set atom, say, $p \subseteq \{X : q(X)\}$ in the head of a set introduction rule $r$ by

$$\text{or}_{\{t:q(t)\in B\}}\, p(t).$$

Let us refer to such a rule as *p-disjunct*. If the head of a $p$-disjunct is empty, remove the rule.

(e) Remove all rules containing a default literal *not* $l$ where $l \in B$, and all remaining occurrences of default literals. □

Consider a program $\Pi$ consisting of facts ($facts(\Pi)$):

```
q(1).      q(2).    q(3).
r(1,1).    r(2,2).
```

and a rule

```
p ⊆ {X:q(X),r(X,X)}.
```

Let $A_1 = facts(\Pi)$. Then $AR(\Pi, A_1)$ consists of facts of $\Pi$ and the rule

```
p(1) or p(2) :- q(1), r(1,1), q(2), r(2,2).
```

We introduce notion $\Pi^R(A)$, called *$\mathscr{A}log$ reduct* wrt $A$, to denote $R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi, A), A)^A$, i.e., the result of applying set introduction reduct, aggregate reduct and the classical reduct to $\Pi$ wrt $A$ in sequence. For a rule $r$, we also use $r^R(A)$ to denote $\{r\}^R(A)$. Similarly, we use $AR(r, B)$ to denote $AR(\{r\}, B)$.

**Lemma 6 (Auxiliary Reduct and Positive Program).** *Let $\Pi$ be stratified by a leveling with maximal ordinal $\lambda$. Then for every $\alpha \leq \lambda$ and every set $B$ of ground regular atoms of levels less then $\alpha$ the auxiliary reduct $AR(\Phi_\alpha, B)$ is a positive program whose heads contain only atoms of level $\alpha$.*

Proof.

1. The steps (a), (c) and (d) of definition of auxiliary reduct of $\Phi_\alpha$ wrt $B$ remove all the set atoms. The step (e) removes all default negations. All new atoms introduced by the reduct are regular. Hence, the program $AR(\Phi_\alpha, B)$ is positive.

2. Consider any rule $r \in AR(\Phi_\alpha, B)$. Let $r'$ be a rule of $\Phi_\alpha$ such that $r = AR(r', B)$.

Since $\Pi$ is stratified, $r'$ is not a constraint. Consider two cases.

Case 1: $r'$ is a proper disjunctive rule. By definition of $\Phi_\alpha$, atoms in the head of $r'$ have level $\alpha$. Since the head of $r'$ is the same as that of $r$, the head of $r$ contains only atoms of level $\alpha$.

Case 2: $r'$ is a set introduction rule defining $p$. Since $r' \in \Phi_\alpha$, all ground atoms formed by $p$ have level $\alpha$. Hence, the head of $r$ contains only atoms of level $\alpha$. □

**Lemma 7 (Consistency of some positive programs).** *A positive program without rules with infinite heads is consistent.*

Proof.[12]
This result follows from the Kuratowski-Zorn Lemma: if every chain in a non-empty poset has a lower bound then the poset contain a minimal element.

Let $\Pi$ be a positive disjunctive program without rules with infinite head. Let $P$ be the poset of all sets of atoms that satisfy the rules of $\Pi$. $P$ is not empty because the se of all atoms of $\Pi$ satisfies its rules.

Let $C$ be a chain in $P$ and let $A$ be the intersection of all elements in $C$. If we show that $A$ satisfies the rules of $\Pi$, then we will have shown that $C$ has a lower bound and the proposition will follow by the Kuratowski-Zorn Lemma.

If $C$ is finite, $A \in C$ and so $A$ satisfies the rules of $\Pi$. Thus, we are done. So, we will now consider the case when $C$ is infinite.

To show that $A$ satisfies rules of $\Pi$ we show that it satisfies every rule in $\Pi$. Consider rule $r$

$$a_1 \text{ or } a_2 \text{ or } \dots \text{ or } a_k \leftarrow b_1, \dots, b_m, \dots$$

and let us assume that $body(r)$ is satisfied by $A$. We need to show that $A$ contains at least one atom $a_i$.

Since $body(r)$ is satisfied by $A$ and $A \subseteq B$, for every element $B$ in the chain $C$, $body(r) \subseteq B$. Since $B$ satisfies rules of $\Pi$, $B$ contains at least one $a_i$. Thus, every $B$ in the chain $C$ contains at least one element $a_i$.

An element $a_i$ is *terminal* if there is a $B_j$ in the chain $C$ such that $a_i \notin B_j$. If all elements $a_i$ in the head of $r$ are terminal, there exist $B_1, \dots, B_k$ such that $a_i \notin B_i$ for

[12] This proof is from Mirek Truszczynski. Minimal edit was made to make it fit the context of this paper.

$i \in 1..k$. No atom in the head of $r$ belongs to $B = B_1 \cap B_2 ... \cap B_k$ (if $a_i$ belongs to $B$ then $a_i$ belongs to $B_i$, a contradiction). But $B$ is an element of the chain $C$ ($B = B_j$, where $B_j$ is the least element of $B_1, \ldots, B_k$ – a well defined set as $B_1, \ldots, B_k$ is a finite chain). This is a contradiction.

Thus, some head atom $a_i$ of $r$ is not terminal. This $a_i$ belongs to all elements in the chain $C$ and so to $A$. □

**Lemma 8 (Consistency of Auxiliary Reducts).** *Let $\Pi$ be stratified by a leveling with maximal ordinal $\lambda$. Then for every $\alpha \leq \lambda$ and every set $B$ of ground regular atoms of levels less then $\alpha$ the auxiliary reduct $AR(\Phi_\alpha, B)$ is consistent, i.e., has an answer set.*

Proof.

Let us denote the auxiliary reduct from the lemma by $T$. By Lemma 6, $T$ is a positive program. Since $\Pi$ is stratified, $\Pi$ has no rules with infinite heads. Such rules can occur in $T$ only by the clause (d) of the definition of the auxiliary reduct. Let $Q$ be a program obtained from $T$ by replacing every rule of $T$ of the form

(*) $p(\bar{t_0})$ or $p(\bar{t_1})$ or $\cdots \leftarrow$ *body*

by

(**) $p(\bar{t_i}) \leftarrow$ *body*

for some $i \geq 0$. Rules of $Q$ contain no infinite heads and hence, by Lemma 7, $Q$ is consistent. Suppose

(1) $A$ is an answer set of $Q$.

We show that

(2) $A$ is an answer set of $T$.

Clearly,

(3) $A$ satisfies rules of $T$.

Consider $X$ such that

(4a) $X \subseteq A$

(4b) $X$ satisfies rules of $T$.

We show that

(5) $X$ satisfies rules of $Q$

(and hence, by (1) and (4a), $A = X$.) Consider

(6) $r \in Q$ such that $body(r)$ is satisfied by $X$.

There are two cases:

(6a) $r$ is of the form (**).

By (4a) and (6) we have that

(7) $body(r)$ is satisfied by $A$.

This, together with (1) implies

(8) $p(\bar{t}_i) \in A$.

The stratification guarantees that no other rule of $Q$ except (**) contains atoms formed by $p$ in the head and hence,

(9) $p(\bar{t}_i)$ is the only atom formed by $p$ which belongs to $A$.

By (4a) and (4b) $p(\bar{t}_i)$ must be in $X$, hence $X$ satisfies $r$.

Now let us look at the second case.

(6b) $r$ is a rule of $T$ with a finite head.

Then, by construction of $Q$

(7) $r \in T$ iff $r \in Q$.

This, together with (4b) implies that $X$ satisfies $r$. Thus, (5) holds, and $A = X$, i.e., no proper subset of $A$ satisfies $T$. Together with (3) this implies (2), which completes the proof of the lemma. □

**Lemma 9 (Auxiliary Reduct and Answer Set).** *Let $\Pi$ be stratified by a leveling $\|\|$ with maximal ordinal $\lambda$. Then, for every $\alpha \leq \lambda$ and every set $B$ of atoms such that for every $a \in B$, $\| a \| < \alpha$ we have that an answer set $A$ of $B \cup AR(\Phi_\alpha, B)$ is also an answer set of $B \cup \Phi_\alpha$.*

Proof.

Let

(1) $A$ be an answer set of $B \cup AR(\Phi_\alpha, B)$ and $S = \{l : \| l \| < \alpha\}$.

Since no atoms from $S$ belong to the heads of rules of $AR(\Phi_\alpha, B)$ supportedness Property1 implies that

(2) $A \cap S = B$.

Let $r$ be an arbitrary rule of $\Phi_\alpha$ and $SE = \{X : cond(X)\}$ be an arbitrary occurrence of a set expression in $r$. For simplicity of presentation we assume that our condition consists of one atom, i.e.,

(3) $SE = \{X : q(X)\}$.

We will show that for every $t$

(4) $q(t) \in B$ iff $q(t) \in A$.

$\Rightarrow$ follows immediately from (2).

To show $\Leftarrow$, let

(5) $q(t) \in A$.

From the definition of $q$ in (3) and the fact that $\|\|$ stratifies $\Pi$ we have that $SE$ is determined by $S$. Therefore, we have two cases:
(a) $q(t)$ is in $S$. Then, by (2) and (5), it is also in $B$.
(b) $q(t)$ has no potential support in $\Pi$. Thus it has no potential support in $B \cup AR(\Phi_\alpha, B)$ either. In this case, by supportedness, $q(t)$ would not be in $A$ which contradicts (5). Thus, (4) holds.

The definition of satisfiability and (4) implies that for ever set atom $SA$ occurring in the body of a rule from $\Phi_\alpha$

(6) $B$ satisfies $SA$ iff $A$ satisfies $SA$. The same is true for $SA$ be undefined.

To prove the conclusion of the proposition we will show that

(7) $A$ satisfies the rules of $B \cup \Phi_\alpha^R(A)$.

By (2) it is sufficient to show that $A$ satisfies $\Phi_\alpha^R(A)$.

Let $r_{alogR}$ be a rule of $\Phi_\alpha^R(A)$ such that

(8) $body(r_{alogR}) \subseteq A$.

We will show that

(9) $A$ satisfies $head(r_{alogR})$.

Let

(10) $r_{phi} \in \Phi_\alpha$ be a rule such that $r_{alogR} \in r_{phi}^R(A)$

Since $r_{alogR} \in \Phi_\alpha^R(A)$ every set atom and every extended literal in the body of $r_{phi}$ is true in $A$ (otherwise, $r_{alogR}$ will be removed by the $\mathscr{A}log$ reduct). Thus, by (6) and (2) we have that

(11) the set atoms and extended literals in $body(r_{phi})$ are true in $B$.

Since $\Pi$ is stratified, $r_{phi}$ is not a constraint. Thus, $r_{phi}$ is a proper disjunctive rule or a set introduction rule.

Case 1: $r_{phi}$ is a proper disjunctive rule. By (11) and definition of auxiliary reduct, $AR(r_{phi}, B)$ is not empty and contains exactly one rule. Furthermore, let $AR(r_{phi}, B)$ be denoted by $r_{ar}$, and we have $head(r_{ar}) = head(r_{alogR})$. By (6) and (11), $body(r_{ar}) = body(r_{alogR})$. Since $A$ is an answer set of $B \cup AR(\Phi_\alpha, B)$ and $r_{ar} \in AR(\Phi_\alpha, B)$, $A$ satisfies $r_{ar}$ and thus $head(r_{ar})$ because of (4). Hence, $A$ satisfies $head(r_{alogR})$.

Case 2: $r_{phi}$ is a set introduction rule with the head $p \subseteq \{X : q(X)\}$. (Other possible heads are treated in a similar manner).

We first prove that

(12) $p \subseteq \{X : q(X)\}$ is true in $A$.

Assume that it is not the case, i.e., there are $p(t)$ and $q(t)$ such that

(13) $p(t) \in A$, and

(14) $q(t) \notin A$.

Consider two cases:

Case 2.1: $AR(r_{phi}, B)$ is empty. Since $r_{phi}$ is the only potential support of $p(t)$ in $\Pi$ by the definition of auxiliary reduct we have that $p(t)$ does not occur in the heads of rules of $B \cup AR(\Phi_\alpha, B)$. It contradicts (1) and (13). The only remaining possibility is

Case 2.2: $AR(r_{phi}, B)$ is not empty. Let us denote it by $r_{ar}$. By clause (3) in the definition of stratification, $r_{phi}$ is the only potential support of $p(t)$ in $\Pi$. This and the definition of auxiliary reduct imply that $r_{ar}$ is the only potential support of $p(t)$ in $AR(\Phi_\alpha, B)$. Since $A$ is an answer set of $B \cup AR(\Phi_\alpha, B)$, (13) and Proposition 1 imply that

(15) $p(t)$ is the only atom of $head(r_{ar})$ that is true in $A$.

By definition of auxiliary reduct, $head(r_{ar})$ is the non-empty disjunction of atoms of the form $p(t_i)$ such that $q(t_i) \in B$. Hence by (15), $p(t)$ is equal to $p(t_i)$ for one of these $i$s. Therefore, $q(t) \in B$, and thus $q(t) \in A$ (because of (2)), contradicting $q(t) \notin A$ (14). There are no other cases and thus (12) is true.

By the definition of $\mathscr{A}log$ reduct, $head(r_{alogR})$ contains at most one atom. In our case, thanks to (10) and (12), $head(r_{alogR})$ is not empty. By definition of set introduction reduct, $head(r_{alogR}) \in A$.

This concludes our proof of (9) and thus (7).

We next show that

(16) $A$ is a minimal set satisfying $B \cup \Phi_\alpha^R(A)$.

Assume it is not the case, i.e.,

(17) there exists $C \subset A$ such that $C$ satisfies $B \cup \Phi_\alpha^R(A)$.

We will show that $C$ satisfies $B \cup AR(\Phi_\alpha, B)$. By (17), it is sufficient to show that

(18) $C$ satisfies $AR(\Phi_\alpha, B)$.

Let $r_{ar}$ be a rule of $AR(\Phi_\alpha, B)$ such that

(19) $body(r_{ar}) \subseteq C$.

We will show

(20) $C$ satisfies $head(r_{ar})$.

Let $r_{phi}$ be a rule of $\Phi_\alpha$ such that $r_{ar} = AR(r_{phi}, B)$. Since $\Pi$ is stratified, $r_{phi}$ is not a constraint, and thus it is either a set introduction rule or a proper disjunctive rule. We prove that $C$ satisfies $head(r)$ by considering those two cases.

Case 1: $r_{phi}$ is a proper disjunctive rule. We first prove some properties on $r_{phi}$ and $A$. Since $r_{ar} \in AR(\Phi_\alpha, B)$, all set atoms and extended literals of form *not* $l$ of $body(r_{phi})$ are true in $B$ by definition of auxiliary reduct. Hence, by (6),

(21) all set atoms of $body(r_{phi})$ are true in $A$.

For every *not* $l \in body(r_{phi})$, $\| l \| < \alpha$ because $r_{phi} \in \Phi_\alpha$ and $\Pi$ is stratified. Hence, $l \in S$ because of (1). Since *not* $l$ is true in $B$, $l \notin B$. Therefore, by (2), $l \notin A$, i.e.,

(22) *not* $l$ is true in $A$.

Since $r_{phi}$ is a proper disjunctive rule, (21) and (22) imply $r_{phi}^R(A)$ is not empty and contains exactly one rule, denoted by $r_{alogR}$. By definition of $\mathscr{A}log$ reduct and auxiliary reduct, $head(r_{alogR}) = head(r_{ar})$. By (4) and (22), $body(r_{alogR}) = body(r_{ar})$, which, together with (19), implies that $C$ satisfies $head(r_{alogR})$ and thus $head(r_{ar})$.

Case 2: $r_{phi}$ is a set introduction rule. Let $head(r_{phi})$ be $p \subseteq \{X : q(X)\}$. Let $r_{ar}$ be of the form:

(23) $or_{\{t:q(t)\in B\}} p(t) \leftarrow body(r_{ar})$.

We first show that

(24) $head(r_{phi})$ is true in $A$.

Assume it is not the case, i.e.,

(25) $head(r_{phi})$ is not true in $A$.

We now show

(26) $r^R_{phi}(A)$ is not empty.

Assume (26) is false. By (21), (22) and the definition of $\mathscr{A}log$ reduct, that $r^R_{phi}(A)$ is empty implies $head(r_{phi})$ is true in $A$. This contradicts (25). Therefore, we have (26). In the proof of case 1 we have already shown that the body of any rule of $r^R_{phi}(A)$ is the same as $body(r_{ar})$. By (25) and definition of set introduction reduct, $r^R_{phi}(A)$ contains a rule of the form $\leftarrow body(r_{ar})$. Since $C$ satisfies $body(r_{ar})$ (19) , it does not satisfy the rule, contradicting that $C$ satisfies $\Phi^R_\alpha(A)$ (17). There, assumption (25) is false and we have (24).

This implies that $r^R_{phi}(A)$ consists of rules

(27) $p(t) \leftarrow body(r_{ar})$ for every $p(t) \in A$.

Let $p(\bar{t}) \in A$. Since $C$ satisfies $r^R_{phi}(A)$ (17) we have that (19) and (27) imply

(28) $p(\bar{t}) \in C$.

Since the head of $r_{ar}$ is true in $A$ and $p(\bar{t}) \in A$,

(29) $q(\bar{t}) \in body(r_{ar})$

by definition of set introduction reduct. Therefore, (19) and (29) imply

(30) $q(\bar{t}) \in C$.

By (2), $C \subset A$ implies $C \cap S \subseteq B$. Hence, (30) implies $q(\bar{t}) \in B$. Together with (28), it implies $C$ satisfies $or_{\{t:q(t)\in B\}} p(t)$, i.e., $head(r_{ar})$ (23).

Therefore, $C$ satisfies $B \cup AR(\Phi_\alpha, B)$. But since, by (18), $C$ is a proper subset of $A$ this contradicts $A$ being an answer set of $B \cup AR(\Phi_\alpha, B)$, and hence (16) is true.

By the definition of answer set, and statements (7) and (16) we have that $A$ is an answer set of $B \cup \Phi_\alpha$, which completes the proof of the lemma. □

**Proposition 4 (Stratification).** *If an $\mathscr{A}log$ program $\Pi$ is stratified then it is consistent.*

Proof of the proposition. Let $\| \ \|$ be a leveling stratifying $\Pi$ with maximal ordinal $\lambda$ and $\Pi_\alpha$ and $\Phi_\alpha$ be as in definition 18. The proof consists of two parts: constructing a sequence $A_\alpha$ of sets of regular atoms of levels not exceeding $\alpha$ and proving that $A_\alpha$ is an answer set of $\Pi_\alpha$. In what follows we use the following notation:

$$A_{<\alpha} =_{def} \begin{cases} A_{\alpha-1} & \text{if } \alpha \text{ is zero or a successor ordinal,} \\ \bigcup_{\beta<\alpha} A_\beta & \text{if } \alpha \text{ is a limit ordinal.} \end{cases}$$

**Part one**. Consider a program

$$\Pi_{<\alpha} =_{def} \{r : r \in \Pi \text{ and } \| r \| < \alpha\}.$$

It is easy to see that by the definition of $\Pi_\alpha$ and $\Phi_\alpha$,

(1) $$\Pi_\alpha = \Pi_{<\alpha} \cup \Phi_\alpha.$$

A sequence $A_\alpha$ is defined using recursion on ordinals. To limit the number of cases in the inductive proof we start with $A_{-1} = \{\}$.

Case 1 of construction: $\alpha$ is 0 or a successor ordinal.
In this case

(1a) $$\Pi_{<\alpha} = \Pi_{\alpha-1}$$

where $\Pi_{-1} = \{\}$. Consider a program $A_{<\alpha} \cup AR(\Phi_\alpha, A_{<\alpha})$. By construction of $A_{<\alpha}$,

(2) the levels of atoms of $A_{<\alpha}$ are less than $\alpha$.

Hence, by Lemma 8, auxiliary reduct $AR(\Phi_\alpha, A_{<\alpha})$ has an answer set. By Lemma 6,

(3) the level of heads of the rules of the auxiliary reduct is $\alpha$.

This implies that the set $S$ of atoms of levels less than $\alpha$ is a splitting set of $A_{<\alpha} \cup AR(\Phi_\alpha, A_{<\alpha})$, and, this, by the splitting set theorem,

(4) program $A_{<\alpha} \cup AR(\Phi_\alpha, A_{<\alpha})$ has an answer set.

Hence, by (2), (3), and the supportedness property of ASP programs, levels of atoms in such answer sets do not exceed $\alpha$. This leads to the following definition. Let

$$A_\alpha \text{ be an answer set of } A_{<\alpha} \cup AR(\Phi_\alpha, A_{<\alpha}).$$

Case 2 of construction: $\alpha$ is a limit ordinal
In this case

(1b) $$\Pi_{<\alpha} = \bigcup_{\beta<\alpha} \Pi_{\beta}.$$

Consider a program $A_{<\alpha} \cup AR(\Phi_{\alpha}, A_{<\alpha})$. As in the previous case we can use the splitting set theorem together with Lemmas 6 and 8 to show that the program has an answer set and that levels of members of these answer sets do not exceed $\alpha$. This time, let

$A_{\alpha}$ be an answer set of $A_{<\alpha} \cup AR(\Phi_{\alpha}, A_{<\alpha})$.

Clearly, by construction, the levels of elements of $A_{\alpha}$ do not exceed $\alpha$. This completes the construction of the sequence.

**Part Two**: We use transfinite induction to show that for every $-1 \leq \alpha \leq \lambda$,

(5a) $A_{\alpha}$ is an answer set of $\Pi_{\alpha}$

and for every $0 \leq \alpha \leq \lambda$,

(5b) $A_{<\alpha}$ is an answer set of $\Pi_{<\alpha}$.

Base. Since $A_{-1}, A_{<0} = \{\}$ and $\Pi_{-1}, \Pi_{<0} = \{\}$ the base is obvious.

Inductive hypothesis: Assume that for every $\beta < \alpha$, $A_{\beta}$ is an answer set of $\Pi_{\beta}$ and $A_{<\beta}$ is an answer set of $\Pi_{<\beta}$.

We first show (5b).

If $\alpha$ is not a limit ordinal this follows immediately from inductive hypothesis, (1a), and the definition of $A_{<\alpha}$.

Suppose now that $\alpha$ is a limit ordinal. By the definition of answer set, (5b) holds iff

(6) $A_{<\alpha}$ is an answer set of $\Pi^{R}_{<\alpha}(A_{<\alpha})$.

By Lemma 5 and (1b) we have that

$$\Pi^{R}_{<\alpha}(A_{<\alpha}) = \bigcup_{\beta<\alpha} \Pi^{R}_{\beta}(A_{<\alpha}).$$

By definition of $\mathscr{A}log$ reduct and clause (1b) of the definition of stratification, for every $\beta < \alpha$, $\Pi^{R}_{\beta}(A_{<\alpha}) = \Pi^{R}_{\beta}(A_{\beta})$ thus

(7) $$\Pi^{R}_{<\alpha}(A_{<\alpha}) = \bigcup_{\beta<\alpha} \Pi^{R}_{\beta}(A_{\beta}).$$

To prove (6) let us first show that

(8) $A_{<\alpha}$ satisfies rules of $\Pi^R_{<\alpha}(A_{<\alpha})$.

Let $r \in \Pi^R_{<\alpha}(A_{<\alpha})$ and assume that $A_{<\alpha}$ satisfies the body of $r$.

Then, by (7), $r \in \Pi^R_{\beta}(A_{\beta})$ for some $\beta < \alpha$. Since, by inductive hypothesis, $A_\beta$ is an answer set of $\Pi^R_{\beta}(A_{\beta})$, we have that $head(r) \in A_\beta$. By definition of $A_{<\alpha}$, $head(r) \in A_{<\alpha}$, which proves (8).

To show minimality assume that there is some proper subset $X$ of $A_{<\alpha}$ which satisfies $\Pi^R_{<\alpha}(A_{<\alpha})$. Since the program is positive, by (7), $X_\beta = \{a : a \in X, \| a \| < \beta\}$ satisfies $\Pi^R_{\beta}(A_{\beta})$. Since $\alpha$ is a limit ordinal by the definition of $A_{<\alpha}$ we have that there is $\beta_0 < \alpha$ such that $X_{\beta_0}$ is a proper subset of $A_{\beta_0}$. But, since by the inductive hypothesis, $A_{\beta_0}$ is an answer set of $\Pi^R_{\beta_0}(A_{\beta_0})$. This is impossible. Thus

$A_{<\alpha}$ is a minimal set satisfying rules of $\Pi^R_{<\alpha}(A_{<\alpha})$.

By definition of answer set, $A_{<\alpha}$ is an answer set of $\Pi^R_{<\alpha}(A_{<\alpha})$ and hence of $\Pi_{<\alpha}$. This proves (8) and hence (6) and (5b).

Now let us prove (5a). Since $\Pi$ is stratified, it is not difficult to check that the set $S$ of atoms of levels less than $\alpha$ is a splitting set of $\Pi_\alpha = \Pi_{<\alpha} \cup \Phi_\alpha$ with $\Pi_{<\alpha}$ being the bottom and $\Phi_\alpha$ being the top. By the definition of $A_{<\alpha}$, $A_\alpha \cap S = A_{<\alpha}$ and hence, by the splitting set theorem (5a) holds iff

(9) $A_{<\alpha}$ is an answer set of $\Pi_{<\alpha}$ and $A_\alpha$ is an answer set of $A_{<\alpha} \cup \Phi_\alpha$.

But (9) holds by (5b) and the definition of $A_\alpha$. This completes the proof of (5a) and of the proposition. □

We need the definition of F-stratification (called aggregate stratification in [27]) and a lemma to prove the result on the relation between $\mathscr{A}log$ and $\mathscr{F}log$ on F-stratified ground programs.

**Definition 20 ($\mathscr{F}log$ Answer Sets).** *Given an $\mathscr{A}\mathscr{F}$-compatible program P and a set A of ground regular atoms we say that*

- *A is a* model *of P if all rules of P are satisfied by A.*
- *The $\mathscr{F}log$* reduct *of P with respect to A, denoted by $R_{\mathscr{F}}(P,A)$, is the program obtained from P by removing every rule whose body contain an element not satisfied by A.*

- *$A$ is an $\mathscr{F}log$* answer set *of $P$ if $A$ is a subset minimal model of $R_{\mathscr{F}}(P,A)$.*

**Definition 21 (F-stratification).** *An $\mathscr{A}\mathscr{F}$-compatible program $P$ is* F-stratified *if there is a level mapping $||\ ||$ from the predicates of $P$ to natural numbers, such that for each rule $r \in P$ and for each prediate $a$ occurring in the head of $r$, the following holds:*

1. *for each predicate $b$ occurring in the body of $r$, $||b|| \leq ||a||$,*
2. *for each predicate $b$ occurring in an aggregate atom of $r$, $||b|| < ||a||$, and*
3. *for each predicate $b$ occurring in the head of $r$, $||b|| = ||a||$.*

**Lemma 10 (Answer Sets of a Program and Its Stratas).** *Given an $\mathscr{A}\mathscr{F}$-compatible program $P$ that is F-stratified with a level mapping, let $P_i$ be the $i^{th}$ strata with respect to the level mapping, $Ha_i$ be the atoms occurring in the head of $P_i$, and $\Pi_i = \cup_{j \leq i} P_j$. For any set $A$ of ground regular atoms, such that $A \subseteq \cup_{j=1}^{\infty} Ha_j$, and $A_i = \cup_{j \leq i}(Ha_j \cap A)$, $A$ is an $\mathscr{A}log$ ($\mathscr{F}log$ respectively) answer set of $P$ iff for any $i$, $A_i$ is an $\mathscr{A}log$ ($\mathscr{F}log$ respectively) answer set of* $\Pi_i$.

Proof.

For any $i$, by definition of $A_i$, we have

(1) $A_i \subseteq A_{i+1}$, and

(2) no atoms of $A_{i+1} \setminus A_i$ occur in $\Pi_i$.

a. $\Longrightarrow$: Assume $A$ is an $\mathscr{A}log$ answer set. We have

(3) $A$ is a minimal model of $R_{\mathscr{A}}(P,A)^A$.

(4) $R_{\mathscr{A}}(P,A)^A = (\cup_{j=1}^{j=i} R_{\mathscr{A}}(P_j,A)^A) \cup (\cup_{j=i+1}^{\infty} R_{\mathscr{A}}(P_j,A)^A)$
(let the latter be denoted by *restPi*)
$= R_{\mathscr{A}}(\Pi_i,A)^A \cup$ *restPi*.

For any $i$, we will show that $A_i$ is an answer set of $\Pi_i$ (17) by showing $A_i$ is a minimal model of $R_{\mathscr{A}}(\Pi_i,A_i)^{A_i}$ (16).

Since $A$ (= $A_i \cup (A \setminus A_i)$) is a model of $R_{\mathscr{A}}(P,A)^A$ and no atoms of $A \setminus A_i$ occur in $\Pi_i$ (by the definition of $A_i$ ), (4) implies

(5) $A_i$ is a model of $R_{\mathscr{A}}(\Pi_i,A)^A$.

We next show $A_i$ is minimal (14) by contradiction. Assume that there exists $B$ such that

(6) $B \subset A_i$, and

(7) $B$ is a model of $R_{\mathscr{A}}(\Pi_i, A)^A$.

For any rule $r \in restPi$, assuming

(8) $B \cup (A \setminus A_i) \models body(r)$,

we prove $B \cup (A \setminus A_i) \models head(r)$ (11).

Since $B \cup (A \setminus A_i) \subseteq A$ and there are no negative atoms or aggregate atoms in $r$, (8) implies

(9) $A \models body(r)$.

Since $A$ is a model of *restPi* by (3) and (4), we have $A \models r$ and thus (9) implies

(10) $A \models head(r)$. Since $head(r)$ does not occur in $\Pi_i$, $A \setminus A_i \models head(r)$, and thus

(11) $B \cup (A \setminus A_i) \models head(r)$. Therefore,

(12) $B \cup (A \setminus A_i) \models restPi$, which, together with (7) and (4), implies

(13) $B \cup (A \setminus A_i) \models R_{\mathscr{A}}(P, A)^A$.

By $B \subset A_i$ (6) and $A_i \subseteq A$, we have $B \cup (A \setminus A_i) \subset A$, which, together with (13), contradicts that $A$ is a minimal model of $R_{\mathscr{A}}(P, A)^A$. Hence,

(14) $A_i$ is a minimal model of $R_{\mathscr{A}}(\Pi_i, A)^A$.

Since no atoms $A \setminus A_i$ occurs in $\Pi_i$, $R_{\mathscr{A}}(\Pi_i, A) = R_{\mathscr{A}}(\Pi_i, A_i)$, and $R_{\mathscr{A}}(\Pi_i, A_i)^A = R_{\mathscr{A}}(\Pi_i, A_i)^{A_i}$. Therefore,

(15) $R_{\mathscr{A}}(\Pi_i, A)^A = R_{\mathscr{A}}(\Pi_i, A_i)^{A_i}$, which, together with (14), implies

(16) $A_i$ is a minimal model of $R_{\mathscr{A}}(\Pi_i, A_i)^{A_i}$, i.e.,

(17) $A_i$ is an $\mathscr{A}log$ answer set of $\Pi_i$.

$\Longleftarrow$:

It can be verified that

(18) $A = \bigcup_{j=1}^{\infty} A_j$.

To show $A$ is an answer set of $\Pi$, we show first show

(19) $A$ is a model of $R_{\mathscr{A}}(\Pi, A)^A$, and

(20) $A$ is mi nimal.

For any $r \in R_{\mathscr{A}}(\Pi,A)^A$, assume

(21) $A$ satisfies $body(r)$.

We will show that $A \models head(r)$.

Let $r' \in R_{\mathscr{A}}(\Pi,A)$ be the rule from which $r$ is obtained and $r'' \in \Pi$ from which $r'$ is obtained. There is $i$ such that $r'' \in \Pi_i$. We know $A_i$ is an answer set of $\Pi_i$.
By definition of aggregate reduct and reduct, (21) implies

(22) $A \models body(r'')$.

Hence, to show $A_i \models body(r'')$, we show

(23) for any $l$ occurring in $body(r'')$, $l \in A_i$ iff $l \in A$.

Since $A_i \subseteq A$, $l \in A_i$ implies $l \in A$. Now we prove the other direction. Assume $l \in A$. By Proposition 1, there must be a rule $r$, at the level at most $i$, supporting $l$. Hence $l \in A \cap \bigcup_{j \leq i} Ha_j$ and thus $l \in A_i$.

By (22) and (23), $A_i \models body(r'')$. Since $A_i$ is an answer set of $\Pi$, $A_i$ satisfies $r''$ and thus, $A_i \models head(r'')$. Therefore, $A_i \models head(r)$ and thus $A \models head(r)$ because $A_i \subseteq A$. As a result, (19) holds.

We next prove (20) by contradiction. Assume

(24) $B \subset A$ is a model of $R_{\mathscr{A}}(\Pi,A)^A$.

We define $B_i = \cup_{j \leq k}(Ha_j \cap B)$. Since $B \subset A$, there must be $k$ such that

(25) $B_k \subset A_k$.

Similar to the proof of (23), one can show

(26) for any $l$ occurring in the body of any rule of $\Pi_k$, $l \in A_k$ iff $l \in A$.

Hence, we can verify

(27) $R_{\mathscr{A}}(\Pi_k,A)^A = R_{\mathscr{A}}(\Pi_k,A_k)^{A_k}$.

Since $\Pi_k \subseteq \Pi$, we have $R_{\mathscr{A}}(\Pi_k,A)^A \subseteq R_{\mathscr{A}}(\Pi,A)^A$. Therefore, (24) and (27) imply that $B$ is a model of $R_{\mathscr{A}}(\Pi_k,A_k)^{A_k}$. We now show $B_k$ is a model of $R_{\mathscr{A}}(\Pi_k,A_k)^{A_k}$. For any rule $r \in R_{\mathscr{A}}(\Pi_k,A_k)^{A_k}$, assume $B_k \models body(r)$. Since there are only positive regular literals in $body(r)$ and $B_k \subseteq B$, $B \models body(r)$. Since $B$ satisfies $r$, $B \models head(r)$. By definition of $B_k$, $B \cap head(r) \subseteq \bigcup_{j=1}^{k}(B \cap Ha_j) = B_k$. Hence, $B_k \models head(r)$. Therefore, $B_k$ is a model of $R_{\mathscr{A}}(\Pi_k,A_k)^{A_k}$, contradicting that $B_k \subset A_k$ and $A_k$ is an answer set of $R_{\mathscr{A}}(\Pi_k,A_k)^{A_k}$. Therefore,

$A$ is an $\mathscr{A}log$ answer set of $P$.

b. $\Longrightarrow$: Assume $A$ is an $\mathscr{F}log$ answer set of $P$. We have

(28) $A$ is a minimal model of $R_{\mathscr{F}}(P,A)$.

(29) $R_{\mathscr{F}}(P,A) = \cup_{j=1}^{j=i} R_{\mathscr{F}}(P_j,A) \cup restPi$, where $restPi = \cup_{j=i+1}^{\infty} R_{\mathscr{F}}(P_j,A)$.

Since no atoms of $A \setminus A_i$ occur in $\Pi_i$, $A = A_i \cup (A \setminus A_i)$ and (28),

(30) $A_i$ is a model of $\cup_{j=1}^{j=i} R_{\mathscr{F}}(P_j,A)$, i.e., $R_{\mathscr{F}}(\Pi_i,A)$.

We prove that $A_i$ is minimal (37) by contradiction. Assume

(31) $B \subset A_i$, and

(32) $B$ is a model of $R_{\mathscr{F}}(\Pi_i,A)$.

For any $r \in restPi$, assuming

(33) $B \cup (A \setminus A_i) \models body(r)$, we prove $B \cup (A \setminus A_i) \models head(r)$ (35).

Since $r \in restPi$, $A \models body(r)$ (by $\mathscr{F}log$ reduct). By (28),

(34) $A \models head(r)$. Since $head(r)$ does not occur in $\Pi_i$, it implies $A \setminus A_i \models head(r)$, and thus

(35) $B \cup (A \setminus A_i) \models head(r)$. Hence,

(36) $B \cup (A \setminus A_i) \models restPi$, which, together with (32) and (29), implies

$B \cup (A \setminus A_i) \models R_{\mathscr{F}}(P,A)$ which, together with $B \cup (A \setminus A_i) \subset A$, contradicts that $A$ is a minimal model of $R_{\mathscr{F}}(P,A)$ (28). Hence,

(37) $A_i$ is a minimal model of $R_{\mathscr{F}}(\Pi_i,A)$.

Since no atoms of $A \setminus A_i$ occurs in $\Pi_i$, $R_{\mathscr{F}}(\Pi_i,A) = R_{\mathscr{F}}(\Pi_i,A_i)$, which, together with (37), implies

$A_i$ is a minimal model of $R_{\mathscr{F}}(\Pi_i,A_i)$. Therefore, $A_i$ is an $\mathscr{F}log$ answer set of $\Pi_i$.

$\Longleftarrow$: Similarly to the proof for the $\mathscr{A}log$ programs, we can show that

$A$ is an $\mathscr{F}log$ answer set of $P$.

$\square$

In the proof below, we use the following notations as defined in the lemma above: $Ha_i$, $P_i$, $\Pi_i$ and $A_i$.

**Proposition 5 ( $\mathscr{A}log$ vs $\mathscr{F}log$ Semantics under F-stratification).** *If an $\mathscr{A}\mathscr{F}$-compatible program P is F-stratified, then A is an $\mathscr{A}log$ answer set of P iff it is an $\mathscr{F}log$ answer set of P.*

Proof.

$\Longleftarrow$: Assuming

(38) $A$ is an $\mathscr{F}log$ answer set of $P$,

we prove $A$ is an $\mathscr{A}log$ answer set of $P$ (79).

By (38) and Lemma 10,

(39) for any $i$, $A_i$ is an $\mathscr{F}log$ answer set of $\Pi_i$.

To prove (79), for any $i(\geq 1)$, we prove $A_i$ is an $\mathscr{A}log$ answer set of $\Pi_i$ by induction on $i$.

Base case: $i = 1$. $A_i$ is an $\mathscr{A}log$ answer set of $\Pi_i$ because $\Pi_i$ contains no aggregate e-atoms.

Inductive hypothesis: for any number $n > 1$, we assume

(40) for any $k$ such that $1 \leq k < n$, $A_k$ is an $\mathscr{A}log$ answer set of $\Pi_k$.

We will prove that $A_n$ is an $\mathscr{A}log$ answer set of $\Pi_n$ (77).

We first prove $A_n$ is a model of $R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$ (52).

For any rule $r'' \in R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$, assuming

(41) $A_n \models body(r'')$,
we prove $A_n \models head(r'')$ (50).

Since $r'' \in R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$, there exists a rule $r \in P$ from which $r''$ is obtained after the aggregate reduct and the classical reduct. Let $r$ be of the form

(42) $head(r)$ :- $posReg(r)$, $negReg(r)$, $aggs(r)$.
*posReg*, *negReg* and *aggs* denotes the regular literals belonging to *r*, the regular literals prefixed with *not* belonging to *r* and the aggregate atoms of *r*.

Since $r'' \in R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$.

(43) $A_n \models aggs(r)$, and

(44) $A_n \models negReg(r)$.

By (42), the form of $r''$ is

(45) $head(r)$ :- $posReg(r)$, $\cup_{agg \in aggs(r)} ta(agg, A_n)$.

By (41),

(46) $A_n \models posReg(r)$, which, together with (44) and (43), implies the existence of rule $r' \in R_{\mathscr{F}}(\Pi_n, A_n)$ which is of the same form as $r$:

(47) $head(r)$ :- $posReg(r)$, $negReg(r)$, $aggs(r)$, and

(48) $A_n \models body(r')$, which, together with (47) and $A_n \models r$ (by (38)), i.e., $A_n \models r'$, implies

(49) $A_n \models head(r')$.

By (45) and (47), $head(r') = head(r'')$. So, (49) implies

(50) $A_n \models head(r'')$, which implies

(51) $A_n \models r''$. Therefore,

(52) $A_n$ is a model of $R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$.

We next show $A_n$ is minimal (76) by contradiction. Assume there exists $B$ such that

(53) $B \subset A_n$, and

(54) $B$ is a model of $R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$.

We note

(55) $A_n \setminus Ha_n = A_{n-1}$ by the definition of $A_n$.

Since $A_n$ is an $\mathscr{F}log$ answer set of $\Pi_n$ (39), (53) implies that there is some rule $r$ of $R_{\mathscr{F}}(\Pi_n, A_n)$ which is not satisfied by $B$, i.e.,

(56) $B \models body(r)$, and

(57) $B \not\models head(r)$.

Since $r \in R_{\mathscr{F}}(\Pi_n, A_n)$, $r \in \Pi_n$. Let $r$ be of the form:

(58) $head(r)$ :- $posReg(r)$, $negReg(r)$, $aggs(r)$.

Since $r \in R_{\mathscr{F}}(\Pi_n, A_n)$,

(59) $A_n \models posReg(r)$,

(60) $A_n \models negReg(r)$, and

(61) $A_n \models aggs(r)$.

(59) to (61) imply that there is a rule $r'' \in R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$ which is obtained from $r$. Rule $r''$ is of the form:

(62) $head(r)$ :- $posReg(r), \cup_{agg \in aggs(r)} ta(agg, A_n)$.

We now prove an intermediate result $A_{n-1} = B \setminus Ha_n$ (68).

Since $B \subset A_n$ (53) and $A_n \setminus Ha_n = A_{n-1}$ (55),

(63) $B \setminus Ha_n \subseteq A_{n-1}$.

By definition of $\Pi_n$ and that $B$ is a model of $R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$ (54),

(64) $B$ is a model of $R_{\mathscr{A}}(\Pi_{n-1}, A_n)^{A_n}$.

Since atoms of $Ha_n$ do not occur in $\Pi_{n-1}$, (64)

(65) $B \setminus Ha_n$ is a model of $R_{\mathscr{A}}(\Pi_{n-1}, A_n)^{A_n}$ = $R_{\mathscr{A}}(\Pi_{n-1}, A_{n-1})^{A_{n-1}}$ (because no atoms of $A_n \setminus A_{n-1}$ occur in $\Pi_{n-1}$).

By induction hypothesis, $A_{n-1}$ is an $\mathscr{A}log$ answer set of $\Pi_{n-1}$. Therefore,

(66) $A_{n-1}$ is a minimal model of $R_{\mathscr{A}}(\Pi_{n-1}, A_{n-1})^{A_{n-1}}$, which, together with (65), implies

(67) $(B \setminus Ha_n) \not\subset A_{n-1}$, which together with (63), implies

(68) $A_{n-1} = B \setminus Ha_n$.

We next prove $B \models \cup_{agg \in aggs(r)} ta(agg, A_n)$ (73).

Since $r \in \Pi_n$, by definition of $Ha_n$, for any $agg \in aggs(r)$, we have

(69) $Base(agg) \cap Ha_n = \{\}$. Therefore,

$$
\begin{aligned}
(70)\ ta(agg, A_n) &= ta(agg, A_n \setminus Ha_n) \\
&= ta(agg, A_{n-1}) \text{ because } A_n \setminus Ha_n = A_{n-1} \text{ (55)} \\
&= ta(agg, B \setminus Ha_n) \text{ because } A_{n-1} = B \setminus Ha_n \text{ (68)} \\
&= ta(agg, B) \text{ by (69).}
\end{aligned}
$$

Hence,

(71) $ta(agg, A_n) = ta(agg, B)$. Since $B \models ta(agg, B)$, we have

(72) $B \models ta(agg, A_n)$. Hence,

(73) $B \models \cup_{agg \in aggs(r)} ta(agg, A_n)$.

By $B \models body(r)$ (56),

(74) $B \models posReg(r)$.

(74) and (73) imply the body of rule $r''$ (62) is satisfied. Since $B$ is a model of $R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$ (54), $B \models r''$. Therefore, $B \models head(r'')$, i.e.,

(75) $B \models head(r)$ because $head(r'') = head(r)$, which contradicts $B \not\models head(r)$ (57). Hence,

(76) $A_n$ is a minimal model of $R_{\mathscr{A}}(\Pi_n, A_n)^{A_n}$. Therefore,

(77) $A_n$ is an $\mathscr{A}log$ answer set of $\Pi_n$. So,

(78) For any $i \geq 1$, $A_i$ is an $\mathscr{A}log$ answer set of $\Pi_i$.

By Lemma 10, (78) implies

(79) $A$ is an $\mathscr{A}log$ answer set of $P$.

$\Longleftarrow$: this is a special case of the results in [41, 42]. □

We next prove the stability result.

**Proposition 6 (Stability of Arithmetics).** *Let $f$ be an aggregate name, $S$ a set expression, $y$ an integer and $\odot$ an arithmetic relation. Program $P_2$ obtained from program $P_1$ by replacing a rule*

$$head \leftarrow body, f(S) \odot y$$

*by*

$$head \leftarrow body, f(S) = Z, Z \odot y.$$

*is strongly equivalent to $P_1$.*

Proof.
We will show that for any program $P$, $P \cup P_1$ and $P \cup P_2$ have the same answer sets, i.e., for any $A$, $A$ is an answer set of $\Pi_1$ iff $A$ is an answer set of $\Pi_2$. Let $\Pi_1 = P \cup P_1$ and $\Pi_2 = P \cup P_2$. Rule $head \leftarrow body, f(S) \odot y$ of $P_1$ is denoted by $r_1$, and $head \leftarrow body, f(S) = Z, Z \odot y$ is denoted by $r_2$.

Consider three cases: $f(S) \odot y$ is *undefined*, *false* and *true* in $A$.

Case 1: $f(S) \odot y$ is *undefined* in $A$. No rule of aggregate reduct $R_{\mathscr{A}}(\Pi_1, A)$ is obtained from $r_1$ because $f(S) \odot y$ is *undefined* in $A$. Similarly, no rule of aggregate reduct $R_{\mathscr{A}}(\Pi_2, A)$ is obtained from $r_2$. Since the $\Pi_1$ and $\Pi_2$ differ only on $r_1$ and $r_2$, $R_{\mathscr{A}}(\Pi_1, A) = R_{\mathscr{A}}(\Pi_2, A)$. Hence, $R_{\mathscr{S}}(R_{\mathscr{A}}(\Pi_1, A), A) = R_{\mathscr{S}}(R_{\mathscr{A}}(\Pi_2, A), A)$ (one can verify that for any program, its answer sets do not depend on the order of applying the aggregate and set introduction reduct). Hence $A$ is an answer set of $\Pi_1$ iff $A$ is an answer set of $\Pi_2$.

Case 2: $f(S) \odot y$ is *false* in $A$. No rule of aggregate reduct $R_{\mathscr{A}}(\Pi_1, A)$ is obtained from $r_1$. If no rule of aggregate reduct $R_{\mathscr{A}}(\Pi_2, A)$ is obtained from (a ground instance of) $r_2$, the proof is the same as Case 1. Assume there is such a rule $r$. It will contain $f(S) = z, z \odot y$. By definition of aggregate reduct, $f(S) = z$ is true. (Otherwise, $r$ does not exist.) Hence, $z \odot y$ is false. (Otherwise, $f(S) \odot y$ is true, contradicting assumption of Case 2.) By definition of set introduction reduct, any rule of $R_{\mathscr{S}}(R_{\mathscr{A}}(\Pi_2, A), A)$ obtained from $r_2$ contains $z \odot y$. Since $z \odot y$ is false, such rule is useless. Let $\Pi' = R_{\mathscr{S}}(R_{\mathscr{A}}(\Pi_2, A), A) \setminus \{r \in R_{\mathscr{S}}(R_{\mathscr{A}}(\Pi_2, A), A) : r$ is obtained from $r_2\}$. One can verify that $\Pi'$ has the same answer sets as $R_{\mathscr{S}}(R_{\mathscr{A}}(\Pi_2, A), A)$. Since $\Pi' = R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi_1, A), A)$, $A$ is an answer set of $\Pi_1$ iff $A$ is an answer set of $\Pi_2$.

Case 3: $f(S) \odot y$ is *true* in $A$. Let $Q_1$ be the set of rules of $R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi_1, A), A)^A$ that are obtained from $r_1$, and $Q_2$ the set of rules of $R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi_2, A), A)^A$ that are obtained from $r_2$. Since $f$ is a function and $f(S) \odot y$ is true in $A$, there is only one $z$ such that $f(S) = z$ and $z \odot y$ is true. One can verify that rules in $Q_1$ are identical to those in $Q_2$ except that the body of the latter contains $z \odot y$ while that of the former does not. One can verify that $Q_1$ is strongly equivalent to $Q_2$. Hence, $R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi_1, A), A)^A$ and $R_{\mathscr{A}}(R_{\mathscr{S}}(\Pi_2, A), A)^A$ have the same answer sets. Hence, $A$ is an answer set of $\Pi_1$ iff $A$ is an answer set of $\Pi_2$.

In summary, we complete the proof. □

**Proposition 7 (Complexity of $\mathscr{A}log$ Programs).** *The problem of checking if a ground atom a belongs to all answer sets of an $\mathscr{A}log$ program is $\Pi_2^P$ complete.*

Proof: Given a ground atom $a$, it is a *cautious consequence* of an $\mathscr{A}log$ program $Q$ if it is true in every answer set of $Q$. We use *cautious reasoning over* $Q$ to denote the problem of checking is $a$ is a cautious consequence of $Q$.

First, cautious reasoning over programs without set atoms is $\Pi_2^P$ hard by [78].

We next show that the cautious reasoning problem for $\mathscr{A}log$ programs belongs to $\Pi_2^P$. For an $\mathscr{A}log$ program $Q$ and a ground atom $a$, the complementary problem is to check if there exists an answer set $S$ of $Q$ such that $a \notin S$. A guess of a set $S$ of literals can be verified with an NP oracle: $Q' = R_{\mathscr{A}}(R_{\mathscr{S}}(Q, S), S)^S$ can be calculated in polynomial time. Testing if $S$ is a minimal model of $Q'$ is in co-NP [79] and hence decidable with one query to an NP oracle. Clearly checking if $a$ is not true in $Q$ is polynomial.

Therefore cautious reasoning over $\mathscr{A}log$ programs is $\Pi_2^P$ complete. □

**Proposition 8 (Complexity of $\mathscr{A}log$ Programs without Disjunctions).** *The problem of checking if a ground literal $l$ belongs to all answer sets of an $\mathscr{A}log$ program without disjunctions is coNP complete.*

Proof. The complementary problem is: given a literal $l$ and a program $\Pi$, checking the existence of an answer set $S$ of $\Pi$ such that $p \notin S$. We will show the complementary problem is in NP. By [45], checking the existing of an answer set of $Q$ without non-aggregate set atoms is NP-complete. We next show that checking a given set a solution of the complementary problem can be done in polynomial time. Let $S$ be a set of literals such that $p \notin S$. $R_{\mathscr{A}}(R_{\mathscr{S}}(Q,S),S)^S$ can be obtained in polynomial time. It is disjunction, negation and aggregate free, and thus its unique answer set $S_1$ can be obtained in polynomial time. The complementary problem can be answered by comparing $S$ and $S_1$ in polynomial time. Hence, the complementary problem is in NP, and thus the proposition holds. □.

**Proposition 9.** *Let* $\Pi$ *be a core* $\mathscr{S}log$ *program. A set A is an* $\mathscr{S}log$ *answer set of* $\Pi$ *iff it is an S-answer set of* $\Pi$.

Proof. For a positive normal logic program $P$ (i.e., a program without *not*, disjunction or set atoms), we use $T_P$ to denote the standard one step fixpoint operator.

$\Longrightarrow$: Since $A$ is an S-answer set of $\Pi$, let $R_{\mathscr{A}}(\Pi,A)$ be an S-reduct of $\Pi$ wrt $A$ such that $A$ is the least fixpoint of $R_{\mathscr{A}}(\Pi,A)^A$. For any aggregate atom occurrence $agg$ in $\Pi$ that is true in $A$, we use $\gamma(agg)$ to denote the regular atoms used to replace $agg$ in the S-reduct.

In the following, for any number $n$, we use $I'_n$ to denote $K_A^\Pi \uparrow n$ and $I''_n$ to denote $T_{R_{\mathscr{A}}(\Pi,A)^A} \uparrow n$.

We first show $I''_n \subseteq I'_n$ by induction on $n$. The base case of $n = 0$ holds. We assume for any $n \geq 1$, $I''_{n-1} \subseteq I'_{n-1}$. We will prove $I''_n \subseteq I'_n$. For any $a \in I''_n$, we will show $a \in I'_n$ (90). Since $a \in I''_n$, there exists a rule $r''$ of $R_{\mathscr{A}}(\Pi,A)^A$ such that $a = head(r'')$ and

(80) $I''_{n-1} \models body(r'')$.

Since $r'' \in R_{\mathscr{A}}(\Pi,A)^A$, there exists $r \in \Pi$ such that $r''$ is obtained from $r$, and we have $A \models aggs(r)$ and

(81) $A \models neg(r)$, which implies that

there exists a rule $r' \in {}^{A}\Pi$ such that $r'$ is obtained from $r$.

For any $agg \in aggs(r)$, we prove $(I'_{n-1}, A) \models agg$ (89), i.e., show $\langle B, Base(agg) \setminus A \rangle$, where $B = I'_{n-1} \cap A \cap Base(agg)$, is an aggregate solution of $agg$. By definition of aggregate solution, for any $S$ such that

(82) $B \subseteq S$ and $S \cap (Base(agg) \setminus A) = \{\}$,

we need show $S \models agg$. Since $\gamma(agg)$ is a minimal guarantee support of $agg$ wrt $A$, $\gamma(agg)$ is a subset of $A$ and of $Base(agg)$. Therefore, $\gamma(agg) \subseteq I''_{n-1} \cap A \cap Base(agg)$ because $\gamma(agg) \subseteq body(r'')$ and (80)). By induction hypothesis, $I''_{n-1} \subseteq I'_{n-1}$, and thus we have

(83) $\gamma(agg) \subseteq I'_{n-1} \cap A \cap Base(agg) = B$.

Since $\gamma(agg)$ is a minimal guarantee support of $agg$ wrt $A$,

(84) for any set $S_1$ such that $\gamma(agg) \subseteq S_1 \subseteq A$, $S_1 \models agg$.

Consider two cases: $S \subseteq A$ and $A \subset S$.

Case 1: $S \subseteq A$. By (82) and (83), $\gamma(agg) \subseteq S$. Therefore, by (84) and $S \subseteq A$, $S \models agg$.

Case 2: $A \subset S$. We show, by contradiction,

(85) $(S \setminus A) \cap Base(agg) = \{\}$.

Assume $(S \setminus A) \cap Base(agg) \neq \{\}$. There exists $x \in (S \setminus A) \cap Base(agg)$, i.e.,

(86) $x \in S$,

(87) $x \notin A$, and

(88) $x \in Base(agg)$.

By (86) and (88), $x \in S \cap Base(agg)$. Therefore, by (87), $S \cap (Base(agg) \setminus A) \neq \{\}$, contradicting (82). By (84), $A \models agg$, which, together with $A \subset S$ and (85), implies that $S \models agg$.

In summary,

(89) for any $agg \in aggs(r)$, $(I'_{n-1}, A) \models agg$.

By (80), $I''_{n-1} \models pos(r)$ and thus $I'_{n-1} \models pos(r)$ because $I''_{n-1} \subseteq I'_{n-1}$. Since $pos(r)$ contains no aggregate atoms, $(I'_{n-1}, A) \models pos(r)$. Together with (89), it implies that $(I'_{n-1}, A) \models body(r')$. Therefore, by definition of $K_A^{\Pi}$,

(90) $a \in I'_n$. Therefore,

(91) $I''_n \subseteq I'_n$ and thus $lfp(T_{R_{\mathscr{A}}(\Pi,A)^A}) \subseteq lfp(K_A^\Pi)$.

Hence,

(92) $A \subseteq lfp(K_A^\Pi)$.

We next show

(93) $K_A^\Pi(A) = A$.

We first show $K_A^\Pi(A) \subseteq A$ (95). For any $a \in K_A^\Pi(A)$, there is a rule $r' \in {}^A\Pi$ such that $a = head(r')$ and

(94) $(A,A) \models body(r')$.

Let $r$ be the rule of $\Pi$ from which $r'$ results. $r' \in {}^A\Pi$ implies $A \models neg(r)$. By (94), $A \models aggs(r)$. Therefore, there is a rule $r'' \in R_{\mathscr{A}}(\Pi,A)^A$ which is obtained from $r$. For all $agg$ occurring in $r$, $\gamma(agg) \subseteq A$. So, (94) implies $A \models body(r'')$ and thus $a \in T_{R_{\mathscr{A}}(\Pi,A)^A}(A) = A$. So,

(95) $K_A^\Pi(A) \subseteq A$.

We next show $A \subseteq K_A^\Pi(A)$ (97). For any $a \in A = T_{R_{\mathscr{A}}(\Pi,A)^A}(A)$, there is a rule $r'' \in R_{\mathscr{A}}(\Pi,A)^A$ such that $a = head(r'')$ and

(96) $A \models body(r'')$.

Let $r$ be the rule of $\Pi$ from which $r''$ results. Since $r'' \in R_{\mathscr{A}}(\Pi,A)^A$, $A \models neg(r)$. Therefore, there is a rule $r' \in {}^A\Pi$ which is obtained from $r$. For any aggregate atom $agg \in aggs(r)$, $A \models agg$ because $r'' \in R_{\mathscr{A}}(\Pi,A)^A$. So, $(A,A) \models agg$. (96) implies $(A,A) \models pos(r')$. Hence, $(A,A) \models body(r')$ and thus $a \in K_A^\Pi(A)$. So,

(97) $A \subseteq K_A^\Pi(A)$.

In summary, (93) holds. Therefore $lfp(K_A^\Pi) \subseteq A$, which, together with (92), implies $A = lfp(K_A^\Pi)$. Therefore, $A$ is an $\mathscr{S}log$ answer set of $\Pi$.

$\Longleftarrow$: Since $A$ is an $\mathscr{S}log$ answer set,

(98) $A = lfp(K_A^\Pi)$.

We now construct an S-reduct of $\Pi$ wrt $A$: $R_{\mathscr{A}}(\Pi,A)$. For any aggregate atom $agg$ occurring in a rule $r$ of $\Pi$ whose body is satisfied by $A$, let $k$ be the least number such that $(I'_k,A) \models agg$. Since $A \models agg$, such $k$ must exist. Let $B = I'_k \cap A \cap Base(agg)$. $\langle B, Base(agg) \setminus A \rangle$ is an aggregate solution of $agg$. Hence, for any $S$ such that $B \subseteq S$ and $S \cap (Base(agg) \setminus A) = \{\}$, $S \models agg$. So, for any $S$ such that $B \subseteq S$ and $S \subseteq A$, $S \models agg$. Therefore, there must be $C \subseteq B$ such that $C$ is a minimal guarantee support of $agg$ wrt $A$. Let $\gamma(agg) = C$. For aggregate atom $agg$ occurring in rules of $\Pi$ whose body contains *not* $l$ and $A \models l$ or $agg$ is not satisfied by $A$, $\gamma(agg)$ is not defined. However, these rules will be removed in producing $R_{\mathscr{A}}(\Pi, A)^A$.

We now show $I'_n = I''_n$ by induction on $n$. The base case of $n = 0$ holds. For any $n \geq 1$, we assume $I'_{n-1} = I''_{n-1}$ and will prove $I'_n = I''_n$ below.

We first show $I'_n \subseteq I''_n$ (101). For any atom $a \in I'_n$, there exists $r' \in {}^A\Pi$ such that $a = head(r')$ and

(99) $(I'_{n-1}, A) \models body(r')$.

Since $r' \in {}^A\Pi$, there exists $r \in \Pi$ such that $r'$ is obtained from $r$ and $A \models neg(r)$. (99) implies $(I'_{n-1}, A) \models aggs(r)$ which in turn implies $A \models aggs(r)$. Therefore, there exists $r'' \in R_{\mathscr{A}}(\Pi, A)^A$ such that $r''$ is obtained from $r$.

By (98) and the monotonicity of $K_A^\Pi$, $I'_{n-1} \subseteq A$. For any $agg \in aggs(r)$, (99) implies $(I'_{n-1}, A) \models agg$, which implies $I'_{n-1} \models agg$. By the construction of $\gamma$ and monotonicity of $K_A^\Pi$, $\gamma(agg) \subseteq I'_{n-1}$. Hence,

(100) $I'_{n-1} \models \gamma(agg)$.

By (99), $I'_{n-1} \models pos(r)$, which, together with (100), implies $I'_{n-1} \models body(r'')$, which, together with the inductive hypothesis $I'_{n-1} = I''_{n-1}$, implies $I''_{n-1} \models body(r'')$. Since $head(r'') = head(r') = a$, by the definition of $I''_n$, $a \in I''_n$. Therefore,

(101) $I'_n \subseteq I''_n$.

Since the proof of $I''_n \subseteq I'_n$ (91) in the necessary condition does not depend on any specific $\gamma$. So, the same proof applies to show $I''_n \subseteq I'_n$ which, together with (101), implies $I'_n = I''_n$. Hence (98) implies that $A$ is the least fixpoint of $R_{\mathscr{A}}(\Pi, A)^A$ too. Therefore, $A$ is an S-answer set of $\Pi$. $\square$