

Well-founded and Stable Semantics of Logic Programs with Aggregates

Nikolay Pelov Marc Denecker
Maurice Bruynooghe

Report CW422, 31 August 2005



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Well-founded and Stable Semantics of Logic Programs with Aggregates*

Nikolay Pelov Marc Denecker
Maurice Bruynooghe

Report CW422, 31 August 2005

Department of Computer Science, K.U.Leuven

In this paper, we present a framework for the semantics and the computation of aggregates in the context of logic programming. In our study, an aggregate can be an arbitrary interpreted second order predicate (or function). We define extensions of the Kripke-Kleene, the well-founded and the stable semantics for aggregate programs. The semantics is based on the concept of a three-valued immediate consequence operator of an aggregate program. Such an operator approximates the standard two-valued immediate consequence operator of the program, and induces a unique Kripke-Kleene model, a unique well-founded model and a collection of stable models. We study different ways of defining such operators and thus obtain a framework of semantics, offering different trade-offs between precision and tractability. In particular, we investigate conditions on the operator that guarantee that the computation of the three types of semantics remains on the same level as for logic programs without aggregates. Other results show that, in practice, even efficient three-valued immediate consequence operators which are very low in the precision hierarchy, still provide optimal precision.

Abstract

Keywords : Logic Programming, Aggregates, Knowledge Representation Logic, Computational Logic.

CR Subject Classification : F.4 Mathematical Logic and Formal Languages, I.2.3 Deduction and Theorem Proving, I.2.4 Knowledge Representation Formalisms and Methods

*Works supported by FWO-Vlaanderen, European Framework 5 Project WASP, and by GOA/2003/08.

1 Introduction

This paper is a study of the semantics of an extension of logic programming with aggregates. Aggregates are specific second order functions or predicates ranging over sets. Standard examples are the minimum of a subset of a partially ordered domain, the cardinality of a set, the sum, the product and the average of a finite set of integers or reals, etc. Aggregates play an important role in different areas. They are used and studied extensively in the context of databases (confer the *group-by* statement). They were introduced in the context of logic programming as declarative variants of the *set_of* and *bag_of* procedures (Kemp and Stuckey 1991; Mumick et al. 1990). Recently, they were introduced in the context of two extensions of logic programming, Answer Set Programming (Simons et al. 2002) and Abductive Logic Programming (Van Nuffelen and Denecker 2000). Aggregates commonly show up in human expert knowledge and expressions of computational problems. For instance:

- the query for the *average* result of students for some exam;
- the property that capacity of a room should exceed the *number* of students attending the course that takes place in that room;
- the *cardinality constraint* that a lecturer should not teach more than 6 courses;
- the property or constraint that the *sum* of the capacities of available power generators in some electricity factory should exceed some given lower bound;

These examples show that aggregates are important to express many forms of human expert knowledge and computational problems in a direct and natural way. For this reason, aggregates likely will be part of computational logics and the languages of future knowledge based systems.

We will study the semantics of sets of rules of the form

$$A \leftarrow \varphi$$

where A is an atomic formula and φ a logic expression possibly containing aggregate formulas. Such rule sets are a core part in logic programming and extensions such as abductive logic programming and extended logic programming, the sub-logic of answer set programming. Rule sets occur also as definitions of intensional predicates in deductive databases and as inductive definitions in the logic *ID-logic*, an extension of classical logic with generalized, non-monotone inductive definitions (Denecker 2000). Thus, the results of our study can be applied in the context of all these logics.

In the context of logic programming, several extensions with aggregates were proposed for subclasses of the formalism that we consider here, in particular for monotone aggregate programs (Mumick et al. 1990; Ross and Sagiv 1997) and stratified aggregate programs (Dell'Armi et al. 2003; Mumick et al. 1990). Our work extends such proposals in two ways. First, we consider more arbitrary rule sets with arbitrary recursion over aggregates. Second, we develop a framework of semantics including extensions of the three main semantics that have been used in the context of the above logics: Kripke-Kleene semantics (i.e. three-valued comple-

tion semantics) (Fitting 1985), stable semantics (Gelfond and Lifschitz 1988) and the well-founded semantics (Van Gelder et al. 1991).

The foundation of our work is the algebraic theory of approximating operators (Denecker et al. 2000; Denecker et al. 2004). Approximation theory is a fixpoint theory of non-monotone lattice operators. With any lattice operator $O: L \rightarrow L$, it associates a family of approximating operators $A: L^2 \rightarrow L^2$ on the product lattice L^2 . The fixpoint theory associates with every approximating operator A different types of fixpoints: a Kripke-Kleene fixpoint and a well-founded fixpoint, both in the bilattice L^2 and a set of A -stable fixpoints of O in the lattice L . In (Denecker et al. 2000) it was shown that the three-valued Fitting operator Φ_P (Fitting 1985) is an approximation of the immediate consequence operator T_P of a logic program P and that the different types of fixpoints of Φ_P corresponds to the Kripke-Kleene, the well-founded and the stable models of P .

In (Denecker et al. 2004), the class of approximations of O was further investigated. The collection of approximations of a lattice operator O is ordered by a precision order. More precise approximations have a more precise Kripke-Kleene and well-founded fixpoint, and have more stable fixpoints. It was shown that O has a most precise approximation, called the ultimate approximation of O which has the most precise semantics.

In the context of logic programming, approximation theory induces a family of Kripke-Kleene, a family of well-founded and a family of stable semantics, generated by the class of approximations of the immediate consequence operator T_P . Basically, each family formalizes the similar intuitions but in different degrees of precision. In (Denecker et al. 2004), the ultimate and the standard versions of these semantics are investigated. It follows from the general theory that the ultimate versions of the semantics are more precise than the standard semantics. Also, ultimate semantics have elegant semantic properties which do not always hold for the standard semantics based on the Fitting operator. For instance, substituting a rule body B by an formula B' which is equivalent with B in classical logic, is equivalence preserving in the ultimate semantics but not in the standard semantics. Also, the ultimate well-founded model of a program with a monotone T_P is the least fixpoint of T_P . On the negative side, applying the ultimate approximation is computationally harder, and it was shown that computing the three types of ultimate semantics for propositional programs is one level higher in the polynomial hierarchy than the standard versions of the same semantics. It was also shown that for important classes of logic programs, standard and ultimate semantics coincide. In fact, it seems that both semantics only differ in the context of programs combining reasoning by cases in rule bodies $((p \wedge \dots) \vee (\neg p \wedge \dots))$ and unstratified recursion over negation over p . In practice, such programs seem to be rare (we are unaware of any practical program with this feature). Thus, the standard semantics based on the Fitting operator and the ultimate semantics based on the ultimate approximation are two very close points in the hierarchy of semantics induced by approximation theory and represent different trade-offs between precision and complexity.

In this work we apply the approximation theory in the context of rule sets with aggregate expressions. We extend the two-valued immediate consequence operator

T_P for aggregate programs, define several different approximating operators of it and study the semantics obtained from them. One operator is the ultimate approximation of T_P . The three types of ultimate semantics obtained from this operator extend the corresponding ultimate semantics for logic programs. They also have the same attractive semantical properties and the high computational complexity. So, we also study extensions of the standard Kripke-Kleene, well-founded and stable semantics of logic programs. To achieve this, we propose the concept of a three-valued aggregate relation approximating a given aggregate relation. We use this concept to define an extension of the Fitting operator Φ_P to the case of programs with aggregates. Since an aggregate relation is approximated by a class of three-valued aggregate relations, we obtain a sub-family of approximations of T_P , all of which coincide with the Fitting operator Φ_P in case P does not contain aggregate expressions. Just as in the case of logic programming without aggregates, the different semantics based on the different approximation operators represent close points in the hierarchy of semantics induced by approximation theory and provide different trade-offs between precision and complexity.

2 Fixpoint Theory of Monotone and Non-monotone Operators

We now present the necessary background on Approximation Theory. For more information we refer to (Denecker et al. 2004).

A structure $\langle L, \leq \rangle$ is a poset if \leq is a reflexive, asymmetric and transitive binary relation on L . A poset $\langle L, \leq \rangle$ is a chain if \leq is a total order, i.e. for each $x, y \in L$, either $x \leq y$ or $y \leq x$.

A poset $\langle L, \leq \rangle$ is chain-complete if each chain $S \subseteq L$ has a least upper bound $\text{lub}(S)$ in L . Since the empty set is a chain, a chain-complete poset has a least element \perp .

A poset $\langle L, \leq \rangle$ is a complete lattice if each subset S of L has a least upper bound $\text{lub}(S)$ and a greatest lower bound $\text{glb}(S)$ in L . In particular, L has a least element \perp and a largest element \top . A complete lattice is chain-complete.

An operator $O: L \rightarrow L$ on a poset $\langle L, \leq \rangle$ is \leq -monotone if for each $x, y \in L$, $x \leq y$ implies $O(x) \leq O(y)$. If $\langle L, \leq \rangle$ is chain-complete, then O has a *least fixpoint* $\text{lfp}(O)$. This fixpoint can be constructively computed as a sequence of *powers* of O defined as follows:

$$\begin{aligned} O \uparrow^0 (x) &= x \\ O \uparrow^{\alpha+1} (x) &= O \uparrow^\alpha (x) \\ O \uparrow^\lambda (x) &= \text{lub}(\{O \uparrow^\alpha (x) \mid \alpha < \lambda\}) \text{ for a limit ordinal } \lambda \end{aligned}$$

Proposition 2.1

If $O: L \rightarrow L$ is a monotone operator and L is a chain-complete poset then there exists an ordinal α such that $O \uparrow^\alpha (\perp) = \text{lfp}(O)$. The least such ordinal is called the *closure ordinal* of O and is denoted with ∞ .

Approximation theory is an extension of the above fixpoint theory to the case

of arbitrary (non-monotone) lattice operators O . The following basic concepts are needed.

Given a complete lattice $\langle L, \leq \rangle$, its *bilattice* is the structure $\langle L^2, \leq, \leq_p \rangle$ where for all $x, y, x', y' \in L$,

$$\begin{aligned} (x, y) \leq (x', y') & \quad \text{if and only if } x \leq x' \text{ and } y \leq y' \\ (x, y) \leq_p (x', y') & \quad \text{if and only if } x \leq x' \text{ and } y' \leq y \end{aligned}$$

The order \leq on L^2 is called the *product order*, while \leq_p is called the *precision order*. Both orders are complete lattice orders on L^2 . We are interested only in a subset of L^2 . A pair (x, y) is *consistent* if $x \leq y$ and *exact* if $x = y$. The set of consistent pairs is denoted L^c .

A basic intuition in approximation theory is that a consistent pair (x, y) approximates an element by a lower and an upper bound. Hence, (x, y) *approximates* any element in the interval $[x, y] = \{z \in L \mid x \leq z \leq y\}$. More precise pairs approximate less elements: $(x, y) \leq_p (x_1, y_1)$ implies $[x, y] \supseteq [x_1, y_1]$. Pairs (x, x) are called exact because they approximate a single element x . The set of exact pairs represents the embedding of L in L^c . The product order is a complete lattice order on L^c while \leq_p is chain-complete on L^c . Hence, any \leq -monotone or \leq_p -monotone operator on L^c has a least fixpoint in the \leq or \leq_p order. Notice also that (\perp, \top) is the \leq_p -least element of L^c while the \leq_p -maximal elements of L^c are precisely the set of exact elements.

Example 2.1

Consider the lattice $TWO = \{\mathbf{f}, \mathbf{t}\}$ of classical truth values ordered as $\mathbf{f} < \mathbf{t}$. We denote the set TWO^c of consistent approximations of TWO with $THREE$.

The set $THREE$ corresponds to the set of truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ used in three-valued logic. The exact pairs (\mathbf{f}, \mathbf{f}) and (\mathbf{t}, \mathbf{t}) , called *false* and *true* correspond to the values \mathbf{f} and \mathbf{t} , while (\mathbf{f}, \mathbf{t}) corresponds to \mathbf{u} , called *undefined*. The product order on $THREE$ corresponds to the truth order $\mathbf{f} < \mathbf{u} < \mathbf{t}$. The precision order on $THREE$ to the order $\mathbf{u} <_p \mathbf{f}, \mathbf{u} <_p \mathbf{t}$, and is sometimes called the knowledge order.

We can define logical connectives in $THREE$ in the following way. Conjunction \wedge and disjunction \vee of two elements are defined as the greatest lower bound and the least upper bound with respect to the product order \leq . The negation operator is defined as $\neg(x, y) = (\neg y, \neg x)$. In particular, $\neg \mathbf{f} = \mathbf{t}$, $\neg \mathbf{t} = \mathbf{f}$, and $\neg \mathbf{u} = \mathbf{u}$. The truth tables of the connectives \wedge , \vee , and \neg are the same as in Kleene's strong three-valued logic.

Definition 2.1 (Approximating Operator)

Let $O: L \rightarrow L$ be an operator on a complete lattice L . We say that $A: L^c \rightarrow L^c$ is an *approximating operator* of O if the following conditions are satisfied:

- A extends O , i.e. $A(x, x) = (O(x), O(x))$ for every $x \in L$;
- A is \leq_p -monotone.

We denote the projections of an approximating operator $A: L^c \rightarrow L^c$ on the first and second components with A^1 and A^2 , i.e. if $A(x, y) = (u, v)$ then $A^1(x, y) = u$ and $A^2(x, y) = v$.

There is a simple and natural intuition behind the concept of an approximating operator A . Any tuple $(x, y) \in L^c$ can be viewed as a non-empty interval $[x, y] = \{z \mid x \leq z \leq y\}$. It is easy to see that for any $z \in [x, y]$, $A(x, y) \leq_p A(z, z) = (O(z), O(z))$ which means that the set $O([x, y])$ is a subset of the interval $A(x, y)$. Hence, $A^1(x, y)$ is a lower estimate and $A^2(x, y)$ is an upper estimate to $O(z)$, for each z in $[x, y]$.

The \leq_p -monotonicity of A guarantees that A has a least fixpoint called the *Kripke-Kleene* fixpoint of A and denoted with $KK(A)$. This fixpoint approximates all fixpoints of O , i.e. if $x = O(x)$ then $KK(A) \leq_p (x, x)$.

Next we define the stable and well-founded fixpoints of A . With a fixed element $b \in L$, we can associate an operator $A^1(\cdot, b)$ mapping any $x \in [\perp, b]$ to $A^1(x, b)$. The operator $A^1(\cdot, b)$ is monotone but as a function from $[\perp, b]$ to L , in general it is not internal in $[\perp, b]$. Similarly, for a fixed element $a \in L$, the operator $A^2(a, \cdot): [a, \top] \rightarrow L$ is monotone, but in general is not internal in $[a, \top]$.

Definition 2.2

The *upper stable operator* $St_A^\uparrow: L \rightarrow L$ maps any $a \in L$ to

$$glb(\{x \in [a, \top]: A^2(a, x) \leq x\}).$$

The *lower stable operator* $St_A^\downarrow: L \rightarrow L$ maps any $b \in L$ to

$$glb(\{x \in [\perp, b]: A^1(x, b) \leq x\}).$$

The upper stable operator maps a lattice element a to the greatest lower bound of the set of pre-fixpoints of $A^2(a, \cdot)$. If this operator is internal in $[a, \top]$ then due to its monotonicity, $St_A^\uparrow(a)$ is its least fixpoint. The upper stable operator maps a lattice element b to the greatest lower bound of the set of pre-fixpoints of $A^1(\cdot, b)$. This set may be empty, in which case $St_A^\downarrow(b) = \top$. However, if $A^1(\cdot, b)$ is internal in $[\perp, b]$, then $St_A^\downarrow(b)$ is its least fixpoint.

Definition 2.3

The *stable revision operator* $St_A: L^2 \rightarrow L^2$ is defined as follows:

$$St_A(a, b) = (St_A^\downarrow(b), St_A^\uparrow(a)).$$

In general, the stable revision operator is not internal in the set L^c . However, there is a subclass of L^c on which this operator has very nice properties. It is defined as the intersection of the following subclasses:

- A pair $(a, b) \in L^c$ is *A-reliable*, if $(a, b) \leq_p A(a, b)$.
- A pair $(a, b) \in L^c$ is *A-prudent*, if $a \leq St_A^\downarrow(b)$.

It is easy to see that if (a, b) is *A-reliable*, then the operators $A^1(\cdot, b)$ and $A^2(a, \cdot)$ are internal in their domain. On the other hand, if (a, b) is *A-prudent*, we can guarantee that a is a safe underestimate of all fixpoints below b of the operator O .

Intuitively, the stable revision operator implements two quite different approximation processes, one to refine the upper estimate b and one to refine the lower estimate a . Given a current upper estimate b , we compute a new lower estimate by an iterative process $x_0 = \perp, x_1 = A^1(x_0, b), \dots, x_{i+1} = A^1(x_i, b), \dots$ until a

fixpoint is reached. In each stage, we use A to approximate $O([x_i, b])$ from below, i.e. by setting $x_{i+1} := A^1(x_i, b) \leq O(z)$, for each $z \in [x_i, b]$. It is easy to see that each computed x_i is a lower estimate to each fixpoint of O below b , and the limit $St_A^\downarrow(b)$ is the best lower bound we can obtain through A to the set of these fixpoints. On the other hand, the refined upper estimate is computed as a limit of the sequence $y_0 = a, y_1 = A^2(a, y_0), \dots, y_{i+1} = A^2(a, y_i), \dots$. The goal is to eliminate non-minimal, *non-reachable* fixpoints of O above the current lower estimate a . Assuming that $a \leq O(a) (= A^2(a, a) = y_1)$, all points in $[a, O(a)]$ are considered reachable. On the next level, also points in $O([a, O(a)])$ above a are of interest, and we can approximate these points from above by computing $A^2(a, y_1) = y_2$. This process is continued until the fixpoint $St_A^\uparrow(a)$ is reached and this fixpoint is taken as the new upper bound.

In (Denecker et al. 2004), it was shown that the set L^{rp} of pairs that are A -reliable and A -prudent contains (\perp, \top) , is chain-complete, and the stable revision operator is an internal, \leq_p -monotone operator in L^{rp} . It follows that this operator has a least fixpoint, called the *well-founded* fixpoint of A and denoted with $WF(A)$. All consistent fixpoints of St_A are A -reliable and A -prudent. They are called *stable fixpoints* of A and they are \leq -minimal fixpoints of A . The subset of exact stable fixpoints is denoted with $ST(A)$. Exact stable fixpoints of A are minimal fixpoints of O (but not vice versa). Exact stable fixpoints can be characterized alternatively as follows: $x \in L$ is an exact stable fixpoint if and only if $O(x) = x$ and $\text{lfp}(A^1(\cdot, x)) = x$.

In general, a lattice operator $O: L \rightarrow L$ may have many approximating operators. For any pair A, B of approximations of O , we define $A \leq_p B$ if and only if for each $(x, y) \in L^c$, $A(x, y) \leq_p B(x, y)$. The following result about the relationship between the different classes of fixpoints of A and B was proven in (Denecker et al. 2004).

Theorem 2.2

If $A \leq_p B$, then $KK(A) \leq_p KK(B)$, $WF(A) \leq_p WF(B)$ and $ST(A) \subseteq ST(B)$.

So, more precise approximating operators lead to more precise Kripke-Kleene and Well-founded fixpoints, and to more exact stable fixpoints. It turns out that O has a most precise approximation U_O called the *ultimate approximation* of O . It is defined as:

$$U_O(x, y) = (\text{glb}(O([x, y])), \text{lub}(O([x, y])))$$

where $O([x, y]) = \{O(z) \mid z \in [x, y]\}$. The Kripke-Kleene, stable and well-founded fixpoints of U_O are called the *ultimate Kripke-Kleene*, *ultimate stable* and *ultimate well-founded fixpoints* of O .

Theorem 2.3 (Denecker et al. 2004)

The ultimate Kripke-Kleene and ultimate well-founded fixpoint of O is the most precise of all Kripke-Kleene and well-founded fixpoints of all approximations of O . The set of ultimate exact stable fixpoints includes all exact stable fixpoints of all approximations A of O .

A special case arises when O is monotone.

Theorem 2.4

If O is monotone, then for every $(x, y) \in L^c$, $U_O(x, y) = (O(x), O(y))$ and its ultimate well-founded fixpoint is exact and is the least fixpoint of O .

In (Denecker et al. 2000), it was shown that the Kripke-Kleene, the well-founded and the stable semantics of a logic program P correspond to Kripke-Kleene, well-founded and exact stable fixpoints of the three-valued immediate consequence operator Φ_P of P defined by Fitting (1985). In (Denecker et al. 2003), analogous results were obtained in the context of default and autoepistemic logic. This shows that approximation theory formalizes an important non-monotonic principle.

3 Aggregates

3.1 Aggregate Functions and Relations

In this text, an aggregate is understood as a second-order n -ary function or relation having at least one set argument. For simplicity, we assume that $n = 1$ in case of aggregate functions and $n = 2$ in case of aggregate relations. We denote the set of all subsets of a set D with $\wp(D)$.

Definition 3.1 (Aggregate Functions and Relations)

Let D_1 and D_2 be domains. An *aggregate function* is any function $F: \wp(D_1) \rightarrow D_2$. An *aggregate relation* is any relation $R \subseteq \wp(D_1) \times D_2$.

We use F to denote an aggregate function and R to denote an aggregate relation. Although many aggregates are functions, for uniformity and convenience sake, our theory below is developed for aggregate relations. If an aggregate function F is used in a context which requires an aggregate relation, F is understood as its *graph* G_F which is the aggregate relation defined as $G_F = \{(S, d) \mid F(S) = d\}$.

We now define a number of standard aggregate functions and relations which we study in this paper. We start with aggregate relations in the context of a poset $\langle D, \leq \rangle$:

Definition 3.2

- $\text{GLB} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid S \in \wp(D) \text{ and } d = \text{GLB}(S)\}$.
- $\text{LUB} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid S \in \wp(D) \text{ and } d = \text{LUB}(S)\}$.
- $\text{LB} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid \forall x \in S, d \leq x\}$.
- $\text{UB} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid \forall x \in S, x \leq d\}$.
- $\text{MIN} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid d \text{ is a minimal element of } S\}$.
- $\text{MAX} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid d \text{ is a maximal element of } S\}$.

The aggregate relations GLB and LUB are (graphs of) partial aggregate functions. If $\langle D, \leq \rangle$ is a complete lattice then GLB and LUB are (graphs of) total aggregate functions. If $\langle D, \leq \rangle$ is a totally ordered set then MIN and MAX represent partial functions. If in addition D is finite then MIN and MAX represent total functions and $\text{MIN} = \text{GLB}$ and $\text{MAX} = \text{LUB}$.

Next, we define aggregate functions on finite sets of numbers. Below, we assume that D is an arbitrary domain and D' is a Cartesian product $D_1 \times \dots \times D_n$ in which D_1 is the set of real numbers \mathbb{R} . Also, we denote the set of all finite subsets of a domain D by $\wp_F(D)$.

Definition 3.3

- CARD: $\wp_F(D) \rightarrow \mathbb{N}$ defined as $\text{CARD}(S) = |S|$, the cardinality of S ;
- SUM: $\wp_F(D') \rightarrow \mathbb{R}$ defined as $\text{SUM}(S) = \sum_{(x_1, \dots, x_n) \in S} x_1$;
- PROD: $\wp_F(D') \rightarrow \mathbb{R}$ defined as $\text{PROD}(S) = \prod_{(x_1, \dots, x_n) \in S} x_1$;
- AVG $\subseteq \wp_F(D') \times \mathbb{R}$ (Average) - the graph of a partial aggregate function defined only for non-empty sets as $(S, d) \in \text{AVG}$ if $d = \text{SUM}(S)/\text{CARD}(S)$.

In the definition of SUM, PROD and AVG, only the first element of a tuple is used to compute the value. The reason to introduce the other arguments is to be able to count one number multiple times. That is, a set $S \subseteq \mathbb{R} \times D_2 \times \dots \times D_n$ represents a *multiset* of real numbers. For example, when counting the total capacity of a building consisting of different rooms, we need to count the capacity of a room as many times as there are rooms with that capacity.

All these aggregate functions have no natural extensions to infinite sets. However, their graphs G_F can be considered as aggregate relations on arbitrary sets — containing only tuples (S, d) for which S is finite.

In this paper, we will focus only on aggregates with one set argument but our theory can be easily extended to the more general case. An example of an aggregate relation with two set arguments is the generalized quantifier $\text{MOST} \subseteq \wp(D) \times \wp(D)$ where $\text{MOST}(A, B)$ expresses that *most A's are B's*. The relation MOST can be easily expressed using the CARD aggregate: $(A, B) \in \text{MOST}$ if and only if $\text{CARD}(A \cap B) > \text{CARD}(A \setminus B)$.

3.2 Derived Aggregate Relations

In this section we show how to obtain new aggregate relations by composition of existing aggregates with other relations.

Definition 3.4

The *composition* of an aggregate relation $R \subseteq \wp(D_1) \times D_2$ with a binary relation $P \subseteq D_2 \times D_3$ is the aggregate relation $R_P \subseteq \wp(D_1) \times D_3$ defined as:

$$R_P = \{(S, d) \mid \exists d' \in D_2: (S, d') \in R \text{ and } (d', d) \in P\}.$$

The *composition* of an aggregate function $F: \wp(D_1) \rightarrow D_2$ with a binary relation $P \subseteq D_2 \times D_3$ is the aggregate relation $F_P \subseteq \wp(D_1) \times D_3$ defined as:

$$F_P = \{(S, d) \mid (F(S), d) \in P\}.$$

Typically, the binary relation P is some partial order relation on the domain D_2 . For example, the CARD_{\geq} aggregate relation is obtained as the composition of the CARD aggregate function with the \geq relation on \mathbb{N} and contains all pairs (S, n) such that $\text{CARD}(S) \geq n$.

An aggregate relation can also be composed with a relation on sets. We consider only one instance of this sort of composition.

Definition 3.5

The *subset aggregate* of an aggregate relation $R \subseteq \wp(D_1) \times D_2$ is the aggregate $R_{\subseteq} \subseteq \wp(D_1) \times D_2$ defined as:

$$R_{\subseteq} = \{(S, d) \mid \exists S': (S', d) \in R \wedge S' \subseteq S\}.$$

For an aggregate function F , F_{\subseteq} denotes the subset aggregate of the graph of F .

3.3 Monotone and Anti-monotone Aggregates

We define two different notions of monotonicity of aggregates, one for functions and one for relations, and then show how they are related.

Definition 3.6

Let $\langle D_2, \leq \rangle$ be a poset and $F: \wp(D_1) \rightarrow D_2$ an aggregate function. We say that F is:

- *monotone* if $S_1 \subseteq S_2$ implies $F(S_1) \leq F(S_2)$;
- *anti-monotone* if $S_1 \subseteq S_2$ implies $F(S_1) \geq F(S_2)$.

The next two propositions list standard aggregate functions which are monotone with respect to some partial order.

Proposition 3.1

Let $\langle D, \leq \rangle$ be a complete lattice. The aggregate function $\text{GLB}: \wp(D) \rightarrow D$ is anti-monotone with respect to \leq and the aggregate function $\text{LUB}: \wp(D) \rightarrow D$ is monotone with respect to \leq .

In Table 1 we use the following notation for subsets of real numbers: \mathbb{R}^+ for the set of non-negative numbers, \mathbb{R}^- for the set of non-positive numbers, $\mathbb{R}^{[1, \infty)}$ for the set of numbers in the interval $[1, \infty)$, and $\mathbb{R}^{(0, 1)}$ for the set of numbers in the interval $[0, 1)$.

Proposition 3.2

Let D be a Cartesian product $D_1 \times \dots \times D_n$ where $n \geq 1$. The aggregate functions in Table 1 from $\wp_F(D)$, the set of finite subsets of D , to the poset $\langle D', \leq \rangle$ are monotone.

aggregate	D_1	$\langle D', \leq \rangle$
CARD	<i>arbitrary</i>	$\langle \mathbb{N}, \leq \rangle$
SUM	\mathbb{R}^+	$\langle \mathbb{R}^+, \leq \rangle$
SUM	\mathbb{R}^-	$\langle \mathbb{R}^-, \geq \rangle$
PROD	$\mathbb{R}^{[1, \infty)}$	$\langle \mathbb{R}^{[1, \infty)}, \leq \rangle$
PROD	$\mathbb{R}^{(0, 1)}$	$\langle \mathbb{R}^{(0, 1)}, \geq \rangle$

Table 1. *Monotone aggregate functions on finite sets*

Monotonicity and anti-monotonicity of an aggregate relation is defined in the following way.

Definition 3.7

Let $R \subseteq \wp(D_1) \times D_2$ be an aggregate relation. We say that R is:

- *monotone* if $(S_1, d) \in R$ and $S_1 \subseteq S_2$ implies $(S_2, d) \in R$;
- *anti-monotone* if $(S_2, d) \in R$ and $S_1 \subseteq S_2$ implies $(S_1, d) \in R$.

The next proposition summarizes the (anti-)monotonicity properties of the aggregate relations defined in Definition 3.2.

Proposition 3.3

The aggregate relations $LB, UB \subseteq \wp(D) \times D$ on a poset $\langle D, \leq \rangle$ are anti-monotone.

All other relations defined in Definition 3.2 are neither monotone nor anti-monotone.

We point out that the graph of a monotone aggregate function is not a monotone aggregate relation according to Definition 3.7. For example, LUB is a monotone aggregate function but its graph is not a monotone aggregate relation. Instead, the composition of an aggregate function F with the inverse of the order with respect to which it is monotone results in a monotone aggregate relation.

Proposition 3.4

Let $F: \wp(D_1) \rightarrow D_2$ be an aggregate function which is monotone with respect to a partial order relation \leq on D_2 . Then F_{\geq} and $F_{>}$ are monotone aggregate relations and F_{\leq} and $F_{<}$ are anti-monotone aggregate relations.

For example, the aggregate relations $CARD_{\geq}$ and $CARD_{>}$ are monotone.

Forming the subset aggregate of any aggregate relation always results in a monotone aggregate relation.

Proposition 3.5

Let $R \subseteq \wp(D_1) \times D_2$ be an arbitrary aggregate relation.

1. R_{\subseteq} is a monotone aggregate relation.
2. If R is a monotone aggregate relation then $R_{\subseteq} = R$.

4 First-order Logic with Aggregates

We introduce aggregates in the context of many-sorted first-order logic.

A *sort symbol* (or simply *sort*) s denotes some sub-domain of the domain of discourse. A *product type* $s_1 \times \cdots \times s_n$ represents the product of the domains represented by the sorts s_1, \dots, s_n and a *set type* $\{s\}$ represents the set of all sets of elements of sort s .

Definition 4.1 (Aggregate Signature)

An *aggregate signature* Σ is a tuple $\langle S; F; P; A \rangle$ where

- S is a set of *sorts*;
- F is a set of sorted *function symbols* $f: s_1 \times \cdots \times s_n \rightarrow w$ where $n \geq 0$;
- P is a set of sorted *predicate symbols* $p: s_1 \times \cdots \times s_n$ where $n \geq 0$;
- A is a set of sorted *aggregate symbols* $R: \{s_1 \times \cdots \times s_n\} \times w$ where $n \geq 1$.

We use $Sort(\Sigma)$, $Func(\Sigma)$, $Pred(\Sigma)$, and $Aggr(\Sigma)$ to denote the sets S , F , P , and A of Σ . We call a function symbol of the form $f: \rightarrow w$ a *constant*. An aggregate symbol $R: \{s_1 \times \dots \times s_n\} \times w$ denotes an aggregate relation between sets of type $s_1 \times \dots \times s_n$ and objects of sort w . Of course, $Aggr(\Sigma)$ may contain many instances of the same type of aggregate relation but with different sorts.

For each sort s , we assume an infinite set V_s of variables of sort s disjoint from the constants in $Func(\Sigma)$. We denote variables, predicate symbols and function symbols with small letters and constants with capital letters.

Definition 4.2 (Terms and atoms)

Let Σ be an aggregate signature. For every sort $s \in S$, we define the set of *terms of type s* by induction:

- a variable $x \in V_s$ of sort s is a term of type s ;
- if $f: s_1 \times \dots \times s_n \rightarrow w \in Func(\Sigma)$ and t_1, \dots, t_n are terms of type s_1, \dots, s_n respectively, then $f(t_1, \dots, t_n)$ is a term of type w .

An *atom* has the form $p(t_1, \dots, t_n)$ where $p: s_1 \times \dots \times s_n \in Pred(\Sigma)$ is a predicate symbol and t_1, \dots, t_n are terms of types s_1, \dots, s_n respectively.

For a fixed aggregate signature Σ we define the notions of set expressions, aggregate atoms and formulas of the logic by simultaneous induction.

Definition 4.3

A *set expression* of type $\{s_1 \times \dots \times s_n\}$ has the form $\{(x_1, \dots, x_n) \mid \varphi\}$ where φ is an aggregate formula called the *condition* of the expression and for each $i = 1, \dots, n$, x_i is a variable of sort s_i .

An *aggregate atom* has the form $R(s, t)$ where $R: \{s_1 \times \dots \times s_n\} \times w \in Aggr(\Sigma)$ is an aggregate symbol, s is a set expression of type $\{s_1 \times \dots \times s_n\}$, and t is a term of type w .

An *aggregate formula* is an atom, an aggregate atom, or an expression of the form $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\forall x\varphi$ and $\exists x\varphi$ where φ and ψ are aggregate formulas and x a variable.

The set of aggregate formulas over Σ is denoted with $\mathcal{L}_\Sigma^{aggr}$.

We illustrate the syntax of aggregate formulas with an example of modeling power plant maintenance.

Example 4.1 (Power Plant Maintenance)

A power plant has a number of power generators called units which have to be scheduled for maintenance. There is a restriction on the total capacity of the units in maintenance. Consider the following aggregate signature:

$$\Sigma = \langle \{u, w, nat\}; \{Max: nat\}; \{capacity: u \times nat, maint: u \times w \times w\}; \\ \{SUM: \{nat \times u\} \times nat\} \rangle.$$

The sort u is interpreted with units and the sort w with weeks. The predicate $capacity(u, c)$ represents that a unit u has a capacity c . The predicate $maint(u, s, e)$ specifies that unit u is in maintenance during the period starting at time point s

(inclusive) and ending at time point e (exclusive). The following aggregate formula expresses that the total capacity of the units in maintenance during a week, should not exceed a value Max :

$$\forall w \text{ SUM}_{\leq}(\{(c, u) \mid \exists s \exists e(\text{maint}(u, s, e) \wedge s \leq w < e \wedge \text{capacity}(u, c))\}, Max).$$

In this formula the sum aggregate computes the sum of all capacities c of units u that are in maintenance during week w . Note that each capacity c is counted as many times as there are units u with capacity c . \square

A *positive literal* is an atom $p(t_1, \dots, t_n)$ and a *negative literal* is the negation of an atom $\neg p(t_1, \dots, t_n)$.

An occurrence of a variable x in an aggregate formula ψ is *bounded* if it occurs in a subformula $\exists x \varphi$ or $\forall x \varphi$ of ψ or in a set expression $\{(x_1, \dots, x, \dots, x_n) \mid \varphi\}$ in ψ . An occurrence of x in ψ is *free* if it is not bounded. The set of free variables of ψ , denoted $Free(\psi)$ is the set of all variables with at least one free occurrence in ψ . Terms and formulas without variables are called *ground* and those without free variables are called *closed*.

Now, we define the semantics of the logic. Let Σ be an aggregate signature.

Definition 4.4 (Structure)

A Σ -*structure* \mathcal{D} consists of the following:

- for each sort $s \in S$ a domain $s^{\mathcal{D}}$;
- for each function symbol $f: s_1 \times \dots \times s_n \rightarrow w \in Func(\Sigma)$ a function

$$f^{\mathcal{D}}: s_1^{\mathcal{D}} \times \dots \times s_n^{\mathcal{D}} \rightarrow w^{\mathcal{D}};$$

- for each predicate symbol $p: s_1 \times \dots \times s_n \in Pred(\Sigma)$ a relation

$$p^{\mathcal{D}} \subseteq s_1^{\mathcal{D}} \times \dots \times s_n^{\mathcal{D}}.$$

- for each aggregate symbol $R: \{s_1 \times \dots \times s_n\} \times w \in Aggr(\Sigma)$ an aggregate relation $R^{\mathcal{D}} \subseteq \wp(s_1^{\mathcal{D}} \times \dots \times s_n^{\mathcal{D}}) \times w^{\mathcal{D}}$.

Consider a Σ -formula $\varphi(x_1, \dots, x_n)$ with free variables x_1, \dots, x_n of sort s_1, \dots, s_n , respectively. Let d_1, \dots, d_n be elements of $s_1^{\mathcal{D}}, \dots, s_n^{\mathcal{D}}$, respectively. Then, $\varphi(d_1, \dots, d_n)$ denotes the formula obtained by substituting d_i for each free occurrence of x_i in φ . So, we consider domain elements as new constants of the respective sorts. We denote this enlarged signature with $\Sigma(\mathcal{D})$ and the corresponding set of formulas with $\mathcal{L}_{\Sigma(\mathcal{D})}^{aggr}$.

Definition 4.5

The value $\llbracket t \rrbracket_{\mathcal{D}}$ of a ground term t for a Σ -structure \mathcal{D} is defined inductively as follows:

- if t is a domain element d , then $\llbracket t \rrbracket_{\mathcal{D}} = d$;
- if t is a constant c , then $\llbracket t \rrbracket_{\mathcal{D}} = c^{\mathcal{D}}$;
- if t is a term $f(t_1, \dots, t_n)$, then $\llbracket t \rrbracket_{\mathcal{D}} = f^{\mathcal{D}}(\llbracket t_1 \rrbracket_{\mathcal{D}}, \dots, \llbracket t_n \rrbracket_{\mathcal{D}})$.

In the following definition and in the rest of the paper we often treat a relation $R \subseteq D$ as a function $R: D \rightarrow \{\mathbf{f}, \mathbf{t}\}$ defined as $R(d) = \mathbf{t}$ if and only if $d \in R$ for an element $d \in D$.

Definition 4.6 (Truth function)

Given is a Σ -structure \mathcal{D} . We define the value of a set expression and the truth value of an aggregate expression by simultaneous induction.

The value $\llbracket \{(x_1, \dots, x_n) \mid \varphi(x_1, \dots, x_n)\} \rrbracket_{\mathcal{D}}$ of a set expression is the set

$$\{(d_1, \dots, d_n) \in s_1^{\mathcal{D}} \times \dots \times s_n^{\mathcal{D}} \mid \mathcal{H}_{\mathcal{D}}(\varphi(d_1, \dots, d_n)) = \mathbf{t}\}$$

The *truth function* $\mathcal{H}_{\mathcal{D}}(\cdot): \mathcal{L}_{\Sigma(\mathcal{D})}^{agg} \rightarrow \mathcal{TW}\mathcal{O}$ for closed aggregate formulas is defined in the following way:

$$\begin{aligned} \mathcal{H}_{\mathcal{D}}(p(t_1, \dots, t_n)) &= p^{\mathcal{D}}(\llbracket t_1 \rrbracket_{\mathcal{D}}, \dots, \llbracket t_n \rrbracket_{\mathcal{D}}) \\ \mathcal{H}_{\mathcal{D}}(\mathbf{R}(\{\bar{x} \mid \varphi\}, t)) &= \mathbf{R}^{\mathcal{D}}(\llbracket \{\bar{x} \mid \varphi\} \rrbracket_{\mathcal{D}}, \llbracket t \rrbracket_{\mathcal{D}}) \\ \mathcal{H}_{\mathcal{D}}(\neg\varphi) &= \neg\mathcal{H}_{\mathcal{D}}(\varphi) \\ \mathcal{H}_{\mathcal{D}}(\varphi \vee \psi) &= \mathcal{H}_{\mathcal{D}}(\varphi) \vee \mathcal{H}_{\mathcal{D}}(\psi) \\ \mathcal{H}_{\mathcal{D}}(\varphi \wedge \psi) &= \mathcal{H}_{\mathcal{D}}(\varphi) \wedge \mathcal{H}_{\mathcal{D}}(\psi) \\ \mathcal{H}_{\mathcal{D}}(\exists x\varphi(x)) &= \bigvee_{d \in s^{\mathcal{D}}} \mathcal{H}_{\mathcal{D}}(\varphi(d)) && \text{(where } s \text{ the sort of } x) \\ \mathcal{H}_{\mathcal{D}}(\forall x\psi(x)) &= \bigwedge_{d \in s^{\mathcal{D}}} \mathcal{H}_{\mathcal{D}}(\psi(d)) && \text{(where } s \text{ the sort of } x) \end{aligned}$$

We define $\mathcal{D} \models \varphi$ if $\mathcal{H}_{\mathcal{D}}(\varphi) = \mathbf{t}$. When $\mathcal{D} \models \varphi$ we call \mathcal{D} a *model* of φ . The relation \models is called the truth relation or the *satisfiability relation*. When the structure \mathcal{D} is clear from the context, we drop the subscript \mathcal{D} of the valuation function $\llbracket \cdot \rrbracket$ and truth function \mathcal{H} .

We illustrate the use of first-order logic with aggregates to formalize the well-known magic square problem.

Example 4.2 (Magic Square)

Given is a $n \times n$ grid which has to be filled with the integer numbers from 1 to n^2 such that the sum of the numbers in all rows, columns, and two diagonals is equal to the same number $M(n)$, known as the *magic constant*:

$$M(n) = \frac{n(n^2 + 1)}{2}$$

Consider the following aggregate signature:

$$\Sigma = \langle \{pos, nat\}; \{Dim: nat, Mc: nat, f: pos \times pos \rightarrow nat\}; \emptyset; \{SUM: \{nat\} \times nat\} \rangle$$

The sort *pos* represents the positions of the table and the sort *nat* the values of the table. The function symbol *f* specifies the number in the corresponding row and column, the constant *Dim* gives the dimension of the grid, and the constant *Mc* gives the magic number. The problem is modeled by the following theory *T*:

$$\begin{aligned} Mc &= Dim * (Dim * Dim + 1) / 2 \\ \forall x \forall y (1 \leq f(x, y) \leq Dim * Dim) \\ \forall x_1 \forall x_2 \forall y_1 \forall y_2 (f(x_1, y_1) = f(x_2, y_2) \rightarrow x_1 = x_2 \wedge y_1 = y_2) \\ \forall y \text{ SUM}(\{z \mid \exists x(z = f(x, y))\}, Mc) \\ \forall x \text{ SUM}(\{z \mid \exists y(z = f(x, y))\}, Mc) \\ \text{SUM}(\{z \mid \exists x(z = f(x, x))\}, Mc) \\ \text{SUM}(\{z \mid \exists x(z = f(x, Dim + 1 - x))\}, Mc) \end{aligned}$$

Consider any structure \mathcal{D} such that $Dim^{\mathcal{D}} = n \in \mathbb{N}$, $pos^{\mathcal{D}} = \{1, \dots, n\}$ and $nat^{\mathcal{D}} = \mathbb{N}$. If \mathcal{D} is a model of T then $f^{\mathcal{D}}$ specifies a solution for the magic square problem of dimension n . \square

5 Aggregate Programs

In this section, we define the syntax of aggregate programs and introduce a basic semantical tool, the T_P operator.

Given is an aggregate signature Σ and a set of sorted predicate symbols Π . We call the symbols from Σ *pre-defined* or *interpreted* while those from Π *defined*. With $\Sigma(\Pi)$ we denote the aggregate signature consisting of both sets of symbols.

From now until the end of this paper, we will assume a fixed aggregate signature Σ and a Σ -structure \mathcal{D} interpreting the pre-defined symbols.

Remark 1

Some of the pre-defined symbols are interpreted on standard domains like:

- sort symbols *nat*, *int*, *real* interpreted by the sets of natural, integer and real numbers respectively;
- the standard function symbols $+$, $*$, $-$, \dots on these sorts interpreted as the corresponding operations on numbers;
- the standard predicate symbols $=$, \leq , \dots on these sorts interpreted as the corresponding relations on numbers;
- all aggregate symbols defined in Section 3.1: *CARD*, *MIN*, *MAX*, *SUM*, \dots

Other interpreted symbols may be domain-specific. In the context of logic programming, the interpretation of the set $S_d \subseteq Sort(\Sigma)$ of domain-specific sorts and the set $F_d \subseteq Func(\Sigma)$ of domain-specific function symbols is normally given by the *free term algebra* generated by F_d . The interpretations $s^{\mathcal{D}}$ of all sorts $s \in S_d$ and the interpretation $f^{\mathcal{D}}$ of all function symbols $f: s_1 \times \dots \times s_n \rightarrow s \in F_d$ are defined by simultaneous induction as follows:

- If $t_1 \in s_1^{\mathcal{D}}, \dots, t_n \in s_n^{\mathcal{D}}$, then $f(t_1, \dots, t_n) \in s^{\mathcal{D}}$.
- If $t_1 \in s_1^{\mathcal{D}}, \dots, t_n \in s_n^{\mathcal{D}}$, then $f^{\mathcal{D}}(t_1, \dots, t_n) = f(t_1, \dots, t_n)$.

In case there is only one domain-specific sort, the free term algebra corresponds to the Herbrand pre-interpretation, i.e. the Herbrand universe and the Herbrand interpretation of function symbols.

The value of domain specific pre-defined predicate symbols may be defined by an extensional database on the domain of \mathcal{D} . \square

A $\Sigma(\Pi)$ -*aggregate rule* r is of the form

$$A \leftarrow \varphi$$

where A is an atom of a defined predicate and φ is a $\Sigma(\Pi)$ -aggregate formula. The atom A is called the *head* of the rule and the formula φ the *body*. We use $body(r)$ to denote the body φ of r . A $\Sigma(\Pi)$ -*aggregate program* is a (possibly infinite) set of

aggregate rules. A *normal aggregate program* is an aggregate program in which the bodies of all rules are conjunctions of literals and aggregate atoms.

Now, we introduce the basic semantical constructs.

The \mathcal{D} -base $base_{\mathcal{D}}(\Pi)$ of Π is defined as

$$base_{\mathcal{D}}(\Pi) = \{p(d_1, \dots, d_n) \mid p: s_1 \times \dots \times s_n \in \Pi, \text{ and} \\ d_1 \in s_1^{\mathcal{D}}, \dots, d_n \in s_n^{\mathcal{D}}\}.$$

The semantics of an aggregate program will be defined in the collection of $\Sigma(\Pi)$ -structures extending \mathcal{D} . For each subset I of $base_{\mathcal{D}}(\Pi)$, we define the $\Sigma(\Pi)$ -structure $\mathcal{D}(I)$ extending \mathcal{D} such that for every atom $A \in base_{\mathcal{D}}(\Pi)$: $\mathcal{H}_{\mathcal{D}(I)}(A) = \mathbf{t}$ if and only if $A \in I$. Clearly, this is a one-to-one correspondence between the subsets of $base_{\mathcal{D}}(\Pi)$ and $\Sigma(\Pi)$ -extensions of \mathcal{D} . In the rest of the paper, we exploit this correspondence and use elements of the power set $\mathcal{I} = \wp(base_{\mathcal{D}}(\Pi))$ to represent $\Sigma(\Pi)$ -extensions of \mathcal{D} . The set \mathcal{I} forms a complete lattice under the subset order \subseteq . This order extends to $\Sigma(\Pi)$ -structures as follows: $\mathcal{D}(I) \leq \mathcal{D}(J)$ if and only if $I \subseteq J$. We will call a subset of $base_{\mathcal{D}}$ an *interpretation*. Equivalently, an interpretation is sometimes also viewed as a mapping from $base_{\mathcal{D}}$ to $\mathcal{TW}\mathcal{O}$.

We introduce the following notation. For any closed defined atom $A = p(t_1, \dots, t_n)$, $\llbracket A \rrbracket_{\mathcal{D}}$ denotes the atom $p(\llbracket t_1 \rrbracket_{\mathcal{D}}, \dots, \llbracket t_n \rrbracket_{\mathcal{D}}) \in base_{\mathcal{D}}(\Pi)$.

Definition 5.1

The *instantiation* of a program P over a structure \mathcal{D} is defined as the set $inst_{\mathcal{D}}(P)$ of all closed rules $A \leftarrow \varphi$ such that:

- there exists a rule $A' \leftarrow \varphi' \in P$ with free variables x_1, \dots, x_m of sorts s_1, \dots, s_m , and
- there exists domain elements $d_1 \in s_1^{\mathcal{D}}, \dots, d_m \in s_m^{\mathcal{D}}$, and
- $A = \llbracket A'(d_1, \dots, d_m) \rrbracket_{\mathcal{D}}$ and
- $\varphi = \varphi'(d_1, \dots, d_m)$.

Note that the body of a rule in the instantiation of an aggregate program is a closed formula containing domain elements.

We now define the two-valued immediate consequence operator of an aggregate program P .

Definition 5.2

The *two-valued immediate consequence operator* $T_{P, \mathcal{D}}^{aggr}: \mathcal{I} \rightarrow \mathcal{I}$ for an aggregate program P is defined as:

$$T_{P, \mathcal{D}}^{aggr}(I) = \{A \mid A \leftarrow \varphi \in inst_{\mathcal{D}}(P) \text{ and } \mathcal{D}(I) \models \varphi\}.$$

This operator extends the T_P operator for normal logic programs defined by van Emden and Kowalski (1976).

Definition 5.3

An interpretation I is a *supported model* of an aggregate program P extending \mathcal{D} if I is a fixpoint of $T_{P, \mathcal{D}}^{aggr}$.

Although the supported model semantics is generally considered to be a weak semantics there are problems with aggregates for which it is the appropriate semantics. One such example is the Party Invitation problem (Ross and Sagiv 1997).

Example 5.1 (Party Invitation I)

A number of people are invited to a party. A person p will accept the invitation if and only if at least k of his (her) friends also accept the invitation. Let us model this problem for the instance in which two friends, say A and B , accept the invitation if and only if the other accepts as well. Consider the following aggregate signature:

$$\Sigma = \langle \{person, nat\}; \{A: person, B: person\}; \{thr: person \times nat, friend: person \times person\}; \{CARD: \{person\} \times nat\} \rangle$$

and let

$$\Pi = \{accept: person\}.$$

Here $friend(x, y)$ means that y is a friend of x and $thr(x, t)$ gives the lower bound t on the number of friends of x . The problem can be modeled by the following single rule:

$$accept(x) \leftarrow thr(x, t) \wedge CARD_{\geq}(\{y \mid friend(x, y) \wedge accept(y)\}, t).$$

Assume that the input structure \mathcal{D} is a Σ -structure in which $person^{\mathcal{D}} = \{A, B\}$ and in which $friend$ and thr is determined by the following table:

$$\begin{array}{ll} friend(A, B). & thr(A, 1). \\ friend(B, A). & thr(B, 1). \end{array}$$

The aggregate program has two supported models in which $accept$ is \emptyset and $\{A, B\}$ respectively. The second solution is not minimal but it is a correct solution to the problem. In reality, A and B may communicate with each other about their decisions to attend the party. \square

Several other examples for which the supported model semantics is appropriate, including an elaborated version of the Party Invitation problem, can be found in (Pelov). It is worth noting that each of these examples can also be expressed in first order logic with aggregates using the completion of the aggregate program.

5.1 Definite Aggregate Programs

In the context of logic programming, definite logic programs are negation free logic programs. A definite program P characterizes a monotone T_P operator and its intended semantics is the least fixpoint of T_P . In this section, we extend the notion of definite program to programs with aggregates.

We define the notions of positive, negative, and neutral aggregate formulas. This definition is not entirely syntactic, but also depends on the monotonicity or anti-monotonicity of aggregate symbols appearing in the formula. We can do that because aggregate symbols always have a fixed interpretation given by the structure \mathcal{D} .

Definition 5.4

An occurrence of a predicate P (resp. a formula ψ) in an aggregate formula φ is *neutral* if it occurs in the condition θ of an aggregate atom $R(\{\bar{x} \mid \theta\}, t)$ in φ such that $R^{\mathcal{D}}$ is neither monotone nor anti-monotone aggregate relation. Otherwise, the

occurrence of P (resp. ψ) is *positive* if the number of negations and aggregate atoms interpreted with anti-monotone aggregate relation above P (resp. ψ) is even. Otherwise, the occurrence of P (resp. ψ) is *negative*.

Definition 5.5 (Positive and Negative Aggregate Formulas)

An aggregate formula φ is *positive* if no defined predicate occurs negatively or neutrally in φ . An aggregate formula φ is *negative* if no defined predicate occurs positively or neutrally in φ .

We note that in the above definition the polarity of pre-defined symbols does not matter. Moreover, if a formula does not contain defined atoms then it is both positive and negative. If the formula φ is an aggregate atom of the form $R(\{\bar{x} \mid \varphi\}, t)$ there are three cases in which it can be positive. The first one is when $R^{\mathcal{D}}$ is a monotone aggregate relation and φ is a positive formula. The second case is when $R^{\mathcal{D}}$ is an anti-monotone aggregate relation and φ is a negative formula. The third one is when $R^{\mathcal{D}}$ is arbitrary and φ does not contain defined predicates. Similarly, the aggregate atom $R(\{\bar{x} \mid \varphi\}, t)$ is negative if $R^{\mathcal{D}}$ is a monotone aggregate relation and φ is negative, $R^{\mathcal{D}}$ is an anti-monotone aggregate relation and φ is positive, or $R^{\mathcal{D}}$ is an arbitrary aggregate relation and φ does not contain defined symbols.

The main property of positive (resp. negative) aggregate formulas is that their satisfiability is monotone (resp. anti-monotone) for a given structure \mathcal{D} .

Proposition 5.1

Let \mathcal{D} be a Σ -structure and ψ be a closed $\Sigma(\Pi)$ -aggregate formula (possibly containing domain elements). For any pair $I \subseteq J \in \text{base}_{\mathcal{D}}(\Pi)$, it holds that:

- if ψ is positive then $\mathcal{D}(I) \models \psi$ implies $\mathcal{D}(J) \models \psi$;
- if ψ is negative then $\mathcal{D}(J) \models \psi$ implies $\mathcal{D}(I) \models \psi$.

Proof

The proof is by induction on the structure of ψ . For positive and negative formulas without aggregates this property is a standard result in the theory of first-order logic. We consider only the case when ψ is an aggregate atom $R(\{\bar{x} \mid \varphi(\bar{x})\}, t)$ without free variables. Let $S_I = \llbracket \{\bar{x} \mid \varphi(\bar{x})\} \rrbracket_{\mathcal{D}(I)}$ and $S_J = \llbracket \{\bar{x} \mid \varphi(\bar{x})\} \rrbracket_{\mathcal{D}(J)}$.

First, let ψ be a positive aggregate atom. We distinguish three cases.

1. $R^{\mathcal{D}}$ is a monotone aggregate relation and $\varphi(\bar{x})$ is a positive formula. For every well-sorted tuple \bar{d} , $\varphi(\bar{d})$ is a positive formula as well. By the induction hypothesis, we have that $\mathcal{D}(I) \models \varphi(\bar{d})$ implies $\mathcal{D}(J) \models \varphi(\bar{d})$. Consequently, $S_I \subseteq S_J$. Finally, because $R^{\mathcal{D}}$ is a monotone aggregate relation, $(S_I, \llbracket t \rrbracket_{\mathcal{D}}) \in R^{\mathcal{D}}$ implies $(S_J, \llbracket t \rrbracket_{\mathcal{D}}) \in R^{\mathcal{D}}$. Thus, $\mathcal{D}(I) \models \psi$ implies $\mathcal{D}(J) \models \psi$.
2. $R^{\mathcal{D}}$ is an anti-monotone aggregate relation and $\varphi(\bar{x})$ is a negative formula. By the induction hypothesis, for every appropriate tuple \bar{d} of domain elements assigned to \bar{x} , we have $\mathcal{D}(J) \models \varphi(\bar{d})$ implies $\mathcal{D}(I) \models \varphi(\bar{d})$. Consequently, $S_J \subseteq S_I$. Finally, because $R^{\mathcal{D}}$ is an anti-monotone aggregate relation, $(S_I, \llbracket t \rrbracket_{\mathcal{D}}) \in R^{\mathcal{D}}$ implies $(S_J, \llbracket t \rrbracket_{\mathcal{D}}) \in R^{\mathcal{D}}$. Thus, $\mathcal{D}(I) \models \psi$ implies $\mathcal{D}(J) \models \psi$.

3. If φ contains no defined predicates, then $\mathcal{D}(J) \models \psi$ if and only if $\mathcal{D}(J) \models \psi$ if and only if $\mathcal{D} \models \psi$.

The proof of anti-monotonicity of negative aggregate atoms is similar and is omitted. \square

We point out that the class of positive (resp. negative) formulas are a strict subset of the class for which the satisfiability relation is monotone (resp. anti-monotone). For example the formula $p \vee \neg p$ is a tautology and hence it is monotone, however it is neither positive nor negative.

Definition 5.6

A *definite aggregate program* is an aggregate program such that the bodies of all rules are positive aggregate formulas.

The class of definite aggregate programs is an extension of the class of definite logic programs and has a monotone immediate consequence operator.

Theorem 5.2

If P is a definite aggregate program then $T_{P,\mathcal{D}}^{aggr}$ is monotone.

Proof

Follows immediately from Proposition 5.1. \square

Definition 5.7

We define the *least fixpoint model* of a definite $\Sigma(\Pi)$ -aggregate program P extending \mathcal{D} as the least fixpoint of its immediate consequence operator $T_{P,\mathcal{D}}^{aggr}$.

A well-known example that can be modeled as a definite aggregate program, is the Company Control problem (Kemp and Stuckey 1991; Mumick et al. 1990; Ross and Sagiv 1997; Van Gelder 1992).

Example 5.2 (Company Controls)

Given is a set of companies which own shares in each other. The problem is to decide if a company x has a controlling interest in a company y . This is the case when x owns (directly or through intermediate companies controlled by x) more than 50% of the stock of y .

To model the problem we use the following aggregate signature:

$$\Sigma = \langle \{c, s\}; \emptyset; \{\text{owns_stock}: c \times c \times s\}; \{\text{SUM}: \{s \times c\} \times s\} \rangle.$$

The sort c represents companies and the sort s represents fractions of shares and is interpreted over the real interval $[0..1]$. The defined predicates are

$$\Pi = \{\text{owns_stock}: c \times c \times s, \text{controls}: c \times c\}.$$

The predicate $\text{owns_stock}(x, y, s)$ means that a company x owns a fraction s of the stock of a company y and $\text{controls}(x, y)$ means that x controls y . The problem is modeled by the aggregate program consisting of the following rule:

$$\text{controls}(x, y) \leftarrow \text{SUM}_{>}(\{(s, z) \mid (x = z \vee \text{controls}(x, z)) \wedge \text{owns_stock}(z, y, s)\}, 0.5).$$

For numbers in the interval $[0..1]$, the SUM aggregate function is monotone with respect to \geq . Consequently, by Proposition 3.4, $\text{SUM}_{>}$ is a monotone aggregate relation. Further, the formula in the aggregate atom is a positive formula, so the aggregate atom in the last rule is monotone. Since none of the bodies contain negation this is a definite aggregate program with a monotone $T_{P,\mathcal{D}}^{\text{agg}}$ operator which has a least fixpoint I . \square

We will now show that the least fixpoint of $T_{P,\mathcal{D}}^{\text{agg}}$ corresponds to the solution to the company control problem. We start by giving a more precise definition of the control relation. Let $sh(a, b)$ be a function which returns the fraction of shares of a company a in a company b or 0 if a does not have shares in b . We define for every $n \in \mathbb{N}$ the *level n control* binary relation, denoted with C^n , by induction on n as follows:

- $C^0 = \emptyset$, i.e. no company has level 0 control of another company;
- $C^{n+1} = \{(a, b) \mid sh(a, b) + \sum_{(a,c) \in C^n} sh(c, b) > 0.5\}$ for $n \geq 0$, i.e., a has a level $n + 1$ control over company b if the sum of the shares of a in b together with the shares of the companies which a has level n control in b is more than 50%.

Clearly, C^n is an increasing sequence of relations. We define the controls relation between companies C as $C = \bigcup_{n \geq 0} C^n$, i.e. a company a controls a company b if, for some $n \geq 0$, $(a, b) \in C^n$.

Proposition 5.3
 $\text{controls}^{\text{lfip}(T_{P,\mathcal{D}}^{\text{agg}})} = C$.

Proof

Let $I_n = T_{P,\mathcal{D}}^{\text{agg}} \uparrow^n (\emptyset)$ for $n \geq 0$. We will prove for each $n \geq 0$ that $\text{controls}^{I_n} = C^n$. Clearly, it follows from this that $\text{controls}^{\text{lfip}(T_{P,\mathcal{D}}^{\text{agg}})} = C$.

For $n = 0$, controls^{I_0} is empty and is equal to C^0 .

For $n > 0$, assume that $\text{controls}^{I_i} = C^i$ for $i = 0, \dots, n - 1$. Fix two companies a and b and consider the value of the instance of the set expression:

$$S = \llbracket \{(s, z) \mid (a = z \vee \text{controls}(a, z)) \wedge \text{owns_stock}(z, b, s)\} \rrbracket_{I_{n-1}}.$$

It is easy to see that if $\text{controls}^{I_{n-1}} = C^{n-1}$ then

$$S = \{(s, c) \mid (a, c) \in C^{n-1} \text{ and } c \text{ contains } s \text{ shares in } b\} \cup S_1$$

where $S_1 = \{(s, a)\}$ if a has s shares in b and $S_1 = \emptyset$ otherwise. It is straightforward then to see that controls^{I_n} contains (a, b) if and only if $(a, b) \in C^n$. \square

Example 5.3 (Borel Sets)

Let \mathbb{R} be the set of real numbers. *Borel sets* are defined by the following monotone inductive definition:

- any open set of real numbers is a Borel set;
- for any countable set C of Borel sets, $\bigcap C$ and $\bigcup C$ are Borel sets;

- if B is a Borel set then $\mathbb{R} - B$ is a Borel set.

To model this definition as an aggregate program consider the following aggregate signature:

$$\Sigma = \langle \{s\}; \{compl: s \rightarrow s\}; \{open: s\}; \{GLB_{\subseteq}^{\omega}, LUB_{\subseteq}^{\omega}: \{s\} \times s\} \rangle.$$

The Σ -structure \mathcal{D} interprets the sort s with the set $\wp(\mathbb{R})$ of all subsets of the real numbers, the predicate *open* is interpreted with the set of open sets, and the function *compl* is interpreted as set complement: $compl^{\mathcal{D}}(S) = \mathbb{R} - S$. The aggregate relations GLB^{ω} and LUB^{ω} are the restrictions of GLB and LUB to countable input sets, i.e. $(S, d) \in GLB^{\omega}$ if and only if $|S| \leq \omega$ and $d = \bigcap S$. The aggregate relations GLB_{\subseteq}^{ω} and LUB_{\subseteq}^{ω} are obtained by forming the subset aggregates of GLB^{ω} and LUB^{ω} respectively (see Definition 3.5). Then $GLB_{\subseteq}^{\omega}(S, d)$ holds if d is the intersection of some countable subset of S . Likewise $LUB_{\subseteq}^{\omega}(S, d)$ holds if d is the union of some countable subset of S .

The program defining Borel sets defines a single defined predicate *borel*: s and contains the following rules:

$$\begin{aligned} borel(S) &\leftarrow open(S). \\ borel(compl(S)) &\leftarrow borel(S). \\ borel(S) &\leftarrow GLB_{\subseteq}^{\omega}(\{B \mid borel(B)\}, S) \vee LUB_{\subseteq}^{\omega}(\{B \mid borel(B)\}, S). \end{aligned}$$

Each of these rules is the formal representation of one of the rules in the inductive definition of Borel sets. Since GLB_{\subseteq}^{ω} and LUB_{\subseteq}^{ω} are monotone aggregate relations (Proposition 3.5) this is a definite aggregate program and it defines a monotone operator $T_{P, \mathcal{D}}^{aggr}$. Consequently, the set of Borel sets is the least set of sets closed under the rules of the inductive definition and this corresponds exactly to the least fixpoint of $T_{P, \mathcal{D}}^{aggr}$. Thus, $borel(d) \in \text{lfp}(T_{P, \mathcal{D}}^{aggr})$ if and only if d is a Borel set. \square

5.2 Stratified Aggregate programs

The important class of stratified aggregate programs was considered already by Mumick et al. (1990) and Dell'Armi et al. (2003). It is a natural extension of the concept of stratified logic program (Apt et al. 1988) where aggregates are treated as negative literals.

An aggregate program P is *stratified* if for each defined predicate p , there is a unique natural number $\|p\| > 0$ called the *level* of p such that if q occurs in the body B of a rule with head p , then $\|q\| \leq \|p\|$ and if q occurs negatively in B or in an aggregate atom, then $\|q\| < \|p\|$. The level $\|P\|$ of P is the maximum of the levels of the defined predicates.

For each level i , let P_i be the set of all rules with a predicate of level i in the head and Π_i the set of defined predicates of level i . Define for each $i \geq 0$, $\Sigma_i = \Sigma \cup \bigcup_{1 \leq j \leq i} \Pi_j$.

Assume $i \geq 1$ and fix a Σ_{i-1} -structure \mathcal{D}' extending \mathcal{D} . Notice that all predicates of Π_i occur only positively in bodies of P_i . Consequently, P_i is a definite aggregate program and has a monotone $T_{P_i, \mathcal{D}'}^{aggr}$ operator. Note that it does not matter whether

the aggregates in Π are monotone or non-monotone, since they do not contain predicates of Π_i .

Definition 5.8

The *standard model* of an aggregate program P extending \mathcal{D} is the interpretation $I = \bigcup_{1 \leq i \leq \|P\|} I_i$ where the set $\{I_i \mid 1 \leq i \leq \|P\|\}$ is defined by the following (finite) induction:

$$\begin{aligned} \mathcal{D}_0 &= \mathcal{D}; \\ I_i &= \text{lfp}(T_{P_i, \mathcal{D}_{i-1}}^{\text{aggr}}); \\ \mathcal{D}_i &= \mathcal{D}(\bigcup_{1 \leq j \leq i} I_j). \end{aligned}$$

The aggregate program in the following example is a stratified aggregate program.

Example 5.4 (Shortest Path)

Consider the signature of directed weighted graphs

$$\Sigma = \langle \{n, w\}; \{edge: n \times n \times w\}; \emptyset; \{\text{MIN}: \{w\} \times w\} \rangle.$$

A Σ -structure \mathcal{D} interprets the sort n with a set of nodes, and the sort w , representing weights, with some set of real numbers $w^{\mathcal{D}} \subseteq \mathbb{R}$. The graph is defined by the relation $edge^{\mathcal{D}}$ where $(a, b, w) \in edge^{\mathcal{D}}$ represents an edge from a to b with weight w .

Consider the following formulation of the problem of finding the weight of the shortest path between two nodes which can be found in (Van Gelder 1992, Example 4.1).

$$sp(x, y, w) \leftarrow \text{MIN}(\{c \mid cp(x, y, c)\}, w).$$

$$cp(x, y, c) \leftarrow edge(x, y, c).$$

$$cp(x, y, c_1 + c_2) \leftarrow cp(x, z, c_1) \wedge edge(z, y, c_2).$$

The aggregate relation MIN is neither monotone nor anti-monotone, so the aggregate atom MIN(...) in the first rule is neutral. Consequently, the program is not definite. However, the program is stratified. The first stratum which defines the $cp/3$ predicate is a definite logic program. The predicate $cp/3$ represents the transitive closure of the graph: $cp(a, b, w)$ is true in the least model of P_{cp} if and only if there is a path between a and b with weight w . The second stratum contains only the definition of $sp/3$ and $sp(a, b, w)$ is true in the standard model of the program if and only if a shortest path between a and b exists and has weight w .

6 Ultimate Semantics for Aggregate Programs

We start our study of the semantics of general aggregate programs with a brief investigation of the semantics generated by the ultimate approximating operator $U_{P, \mathcal{D}}^{\text{aggr}}$ of $T_{P, \mathcal{D}}^{\text{aggr}}$. This semantics of aggregate programs was first studied by Denecker et al. (2001).

Definition 6.1

The *ultimate approximating operator* $U_{P,\mathcal{D}}^{aggr} : \mathcal{I}^c \rightarrow \mathcal{I}^c$ of $T_{P,\mathcal{D}}^{aggr} : \mathcal{I} \rightarrow \mathcal{I}$ is defined as:

$$U_P^{aggr}(I_1, I_2) = \left(\bigcap_{I \in [I_1, I_2]} T_{P,\mathcal{D}}^{aggr}(I), \bigcup_{I \in [I_1, I_2]} T_{P,\mathcal{D}}^{aggr}(I) \right).$$

Definition 6.2

The *ultimate Kripke-Kleene model*, the *ultimate well-founded model*, and the set of *ultimate stable models* of an aggregate program P are defined as the Kripke-Kleene, the well-founded, and the set of exact stable fixpoints of the $U_{P,\mathcal{D}}^{aggr}$ operator.

We obtain the following corollary to Theorem 2.4 and Theorem 5.2.

Corollary 6.1

If $T_{P,\mathcal{D}}^{aggr}$ is monotone, then the ultimate well-founded fixpoint of P is the least fixpoint of $T_{P,\mathcal{D}}^{aggr}$ and the unique ultimate stable fixpoint of P . If P is a definite aggregate program, then its ultimate well-founded model and unique ultimate stable model is the least fixpoint model of P .

It follows that both the ultimate well-founded and the ultimate stable semantics correctly model the company control program in Example 5.2 and the Borel sets program in Example 5.3. Later, we will also show a similar result to Corollary 6.1 for stratified programs: if P is a stratified program then its ultimate well-founded and unique ultimate stable model coincide with the standard model of P . Hence, the ultimate semantics also models correctly the shortest path program in Example 5.4.

Two aggregate programs with the same immediate consequence operator are equivalent under ultimate semantics. Since substituting formulas in rule bodies by equivalent formulas preserves the operator, this operation is equivalence preserving.

Proposition 6.2

Let P and P' be aggregate programs such that P' is obtained by substituting a formula φ' for a formula φ in the body of a rule of P . If $\forall(\varphi \equiv \varphi')$ is satisfied in all $\Sigma(\Pi)$ -extensions of \mathcal{D} , then P and P' are equivalent under the ultimate Kripke-Kleene, well-founded and stable semantics.

The nice semantical properties of ultimate semantics come at a computational price. Even for programs without aggregates, computing the ultimate well-founded model is co-NP-hard and deciding the existence of a two-valued ultimate stable model is Σ_2^P -complete (Denecker et al. 2004). For this reason, we will study weaker semantics based on less precise approximations of $T_{P,\mathcal{D}}^{aggr}$.

7 Extending the Standard Well-founded and Stable Semantics

The goal of this section is to extend the Kripke-Kleene (Fitting 1985), well-founded (Van Gelder et al. 1991) and stable (Gelfond and Lifschitz 1988) semantics of normal logic programming. According to Approximation Theory (Denecker et al. 2000) these three semantics can be obtained from the three-valued immediate consequence

operator Φ_P defined by Fitting (1985). In particular, the collection of three-valued interpretations corresponds to the set L^c of consistent pairs of the lattice L of two-valued interpretations. The operator Φ_P is an approximation on this set, it approximates the T_P operator and its stable and well-founded fixpoints correspond to the stable and well-founded models of P . By extending Φ_P to the class of aggregate programs, we will be able to obtain well-founded and stable semantics which extend those of logic programs without aggregates.

To extend the Fitting operator for aggregate programs, we must be able to evaluate the aggregate formulas in three-valued interpretations. For this reason we introduce the concept of a three-valued structure. It is similar to standard structures, except that predicates are assigned three-valued relations and aggregate symbols are assigned three-valued aggregate relations. Because the value of a set expression in a three-valued structure can be a three-valued set, three-valued aggregates take three-valued sets as argument. Below, we formalize these notions. In the sequel, it will be convenient to view an aggregate relation both as a subset of $\wp(D_1) \times D_2$ and as a function from $\wp(D_1) \times D_2$ to $\mathcal{TW}\mathcal{O}$.

Definition 7.1 (Three-valued Aggregate Relations)

A *three-valued aggregate relation* is a function $\mathcal{R}: \wp(D_1)^c \times D_2 \rightarrow \mathcal{THRE}\mathcal{E}$ which satisfies:

- *\leq_p -monotonicity*: for every pair of three-valued sets $\tilde{S}_1, \tilde{S}_2 \in \wp(D_1)^c$ and for every $d \in D_2$, if $\tilde{S}_1 \leq_p \tilde{S}_2$ then $\mathcal{R}(\tilde{S}_1, d) \leq_p \mathcal{R}(\tilde{S}_2, d)$;
- *exactness*: for every exact (two-valued) set $S \in \wp(D_1)$ and for every $d \in D_2$, $\mathcal{R}((S, S), d) \in \mathcal{TW}\mathcal{O}$.

The concept of a three-valued aggregate relation is very similar to approximating operators (Definition 2.1).

Remark 2

The definition has a straightforward extension to aggregates with multiple set arguments by requiring \leq_p -monotonicity and exactness conditions for all set arguments.

A three-valued aggregate relation \mathcal{R} *approximates* an aggregate relation R if for each set $S \in \wp(D_1)$ and for each $d \in D_2$, $\mathcal{R}((S, S), d) = R(S, d)$. Due to the exactness condition, a three-valued aggregate relation approximates exactly one aggregate relation.

Recall that $\mathcal{THRE}\mathcal{E} = \mathcal{TW}\mathcal{O}^c$. It follows that a three-valued aggregate relation $\mathcal{R}: \wp(D_1)^c \times D_2 \rightarrow \mathcal{THRE}\mathcal{E}$ is completely determined by the pair $(\mathcal{R}^1, \mathcal{R}^2)$ of its projections on the first and second component. These projections are relations $\mathcal{R}^1, \mathcal{R}^2 \subseteq \wp(D_1)^c \times D_2$ such that $\mathcal{R}^1 \subseteq \mathcal{R}^2$ and $\mathcal{R}^1, \mathcal{R}^2$ coincide on two-valued sets¹. We will frequently define an three-valued aggregate relation \mathcal{R} by defining \mathcal{R}^1 and \mathcal{R}^2 separately.

¹ Equivalently, $\mathcal{R}^1, \mathcal{R}^2$ are functions $\wp(D_1)^c \times D_2 \rightarrow \mathcal{TW}\mathcal{O}$ which coincide on two-valued sets and such that $\mathcal{R}^1 \leq \mathcal{R}^2$.

Definition 7.2 (Three-valued Structure)

Let Σ be an aggregate signature. A *three-valued Σ -structure* $\tilde{\mathcal{D}}$ consists of the following:

- for each sort $s \in S$ a domain $s^{\tilde{\mathcal{D}}}$;
- for each function symbol $f: s_1 \times \cdots \times s_n \rightarrow w \in \text{Func}(\Sigma)$ a function

$$f^{\tilde{\mathcal{D}}}: s_1^{\tilde{\mathcal{D}}} \times \cdots \times s_n^{\tilde{\mathcal{D}}} \rightarrow w^{\tilde{\mathcal{D}}};$$

- for each predicate symbol $p: s_1 \times \cdots \times s_n \in \text{Pred}(\Sigma)$ a *three-valued relation*

$$p^{\tilde{\mathcal{D}}}: s_1^{\tilde{\mathcal{D}}} \times \cdots \times s_n^{\tilde{\mathcal{D}}} \rightarrow \text{THRE}.$$

- for each aggregate symbol $R: \{s_1 \times \cdots \times s_n\} \times w \in \text{Aggr}(\Sigma)$ a *three-valued aggregate relation*

$$R^{\tilde{\mathcal{D}}}: \wp((s_1^{\tilde{\mathcal{D}}})^c \times \cdots \times (s_n^{\tilde{\mathcal{D}}})^c) \times w^{\tilde{\mathcal{D}}} \rightarrow \text{THRE}.$$

A three-valued Σ -structure $\tilde{\mathcal{D}}$ *approximates* a Σ -structure \mathcal{D} if for each predicate symbol p , $p^{\tilde{\mathcal{D}}}$ approximates $p^{\mathcal{D}}$ and for each aggregate symbol R , $R^{\tilde{\mathcal{D}}}$ approximates $R^{\mathcal{D}}$.

Now we define a precision order between three-valued aggregate relations and structures.

Definition 7.3

For all three-valued aggregate relations $\mathcal{R}_1, \mathcal{R}_2: \wp(D_1)^c \times D_2 \rightarrow \text{THRE}$, define $\mathcal{R}_1 \leq_p \mathcal{R}_2$ if $\mathcal{R}_1(\tilde{S}, d) \leq_p \mathcal{R}_2(\tilde{S}, d)$ for every three-valued set $\tilde{S} \in \wp(D_1)^c$ and domain element $d \in D_2$.

For all three-valued Σ -structures $\tilde{\mathcal{D}}_1$ and $\tilde{\mathcal{D}}_2$, define $\tilde{\mathcal{D}}_1 \leq_p \tilde{\mathcal{D}}_2$ if $\tilde{\mathcal{D}}_1$ and $\tilde{\mathcal{D}}_2$ have the same domain, the same interpretations of sort and function symbols, for each predicate symbol $p \in \text{Pred}(\Sigma)$, $p^{\tilde{\mathcal{D}}_1} \leq_p p^{\tilde{\mathcal{D}}_2}$ and for each aggregate symbol $R \in \text{Aggr}(\Sigma)$, $R^{\tilde{\mathcal{D}}_1} \leq_p R^{\tilde{\mathcal{D}}_2}$.

It is straightforward to see that if $\mathcal{R}_1 \leq_p \mathcal{R}_2$ and \mathcal{R}_2 approximates an aggregate relation R then \mathcal{R}_1 also approximates R .

Definition 7.4 (Three-valued valuation and truth functions)

Let Σ be an aggregate signature and $\tilde{\mathcal{D}}$ be a three-valued Σ -structure. We define the *three-valued valuation function* $\llbracket \cdot \rrbracket_{\tilde{\mathcal{D}}}$ for set expressions and the *three-valued truth function* $\mathcal{H}_{\tilde{\mathcal{D}}}$ for aggregate formulas by simultaneous induction.

Let $\{(x_1, \dots, x_n) \mid \varphi(x_1, \dots, x_n)\}$ be a set expression of type $\{s_1 \times \cdots \times s_n\}$. The value $\llbracket \{(x_1, \dots, x_n) \mid \varphi\} \rrbracket_{\tilde{\mathcal{D}}}$ is the three-valued set \tilde{S} defined as:

$$\tilde{S}(d_1, \dots, d_n) = \mathcal{H}_{\tilde{\mathcal{D}}}(\varphi(d_1, \dots, d_n))$$

for every $(d_1, \dots, d_n) \in s_1^{\tilde{\mathcal{D}}} \times \cdots \times s_n^{\tilde{\mathcal{D}}}$.

The *three-valued truth function* for first-order aggregate formulas $\mathcal{H}_{\tilde{\mathcal{D}}}(\cdot): \mathcal{L}_{\Sigma(\tilde{\mathcal{D}})}^{aggr} \rightarrow \text{THRE}$ is defined as in Definition 4.6 using the three-valued operations \wedge , \vee , and \neg as defined in Example 2.1.

Next, we show that the three-valued truth function $\mathcal{H}_{\tilde{\mathcal{D}}}$ is monotone with respect to the precision order \leq_p on three-valued interpretations.

Proposition 7.1

Let $\tilde{\mathcal{D}}_1$ and $\tilde{\mathcal{D}}_2$ be three-valued Σ -structures. If $\tilde{\mathcal{D}}_1 \leq_p \tilde{\mathcal{D}}_2$ then for every Σ -aggregate formula φ , $\mathcal{H}_{\tilde{\mathcal{D}}_1}(\varphi) \leq_p \mathcal{H}_{\tilde{\mathcal{D}}_2}(\varphi)$.

Proof

The proof is by a standard induction argument on the size of φ . For aggregate atoms it follows from the \leq_p -monotonicity of the three-valued aggregate relations. \square

Another proposition shows the correspondence between three-valued and two-valued truth functions. If all predicate symbols have two-valued interpretations then evaluating an aggregate formula in a three-valued structure results in a two-valued truth value.

Proposition 7.2

Let \mathcal{D} be a Σ -structure and $\tilde{\mathcal{D}}$ be a three-valued structure which approximates \mathcal{D} . For every aggregate formula φ such that $p^{\tilde{\mathcal{D}}}$ is two-valued for all predicates p appearing in φ , $\mathcal{H}_{\tilde{\mathcal{D}}}(\varphi) = (\mathcal{H}_{\mathcal{D}}(\varphi), \mathcal{H}_{\mathcal{D}}(\varphi))$.

Proof

The proof is by a standard induction argument on the size of φ . For aggregate atoms it follows from the exactness condition of the three-valued aggregate relations. \square

In the sequel we will consider only three-valued structures for which only the interpretation of the defined predicates Π and the aggregates is three-valued while the interpretation of the pre-defined predicates is two-valued. Such structures are denoted with $\tilde{\mathcal{D}}(\tilde{I})$ where $\tilde{I}: base_{\tilde{\mathcal{D}}}(\Pi) \rightarrow \mathcal{T}\mathcal{H}\mathcal{R}\mathcal{E}\mathcal{E}$ gives the (three-valued) interpretation of the predicates in Π .

We now define a three-valued operator $\Phi_{P, \tilde{\mathcal{D}}}^{aggr}$ for aggregate programs as follows.

Definition 7.5

The *three-valued immediate consequence operator* $\Phi_{P, \tilde{\mathcal{D}}}^{aggr}: \mathcal{I}^c \rightarrow \mathcal{I}^c$ for an aggregate program P is defined as:

$$\Phi_{P, \tilde{\mathcal{D}}}^{aggr}(\tilde{I})(A) = \bigvee \{ \mathcal{H}_{\tilde{\mathcal{D}}(\tilde{I})}(\varphi) \mid A \leftarrow \varphi \in inst(P) \}.$$

The least upper bound \bigvee in the definition of $\Phi_{P, \tilde{\mathcal{D}}}^{aggr}$ is taken with respect to the standard truth order on $\mathcal{T}\mathcal{H}\mathcal{R}\mathcal{E}\mathcal{E}$ and corresponds to a disjunction.

Proposition 7.3

If $\tilde{\mathcal{D}}$ is a three-valued structure approximating \mathcal{D} then $\Phi_{P, \tilde{\mathcal{D}}}^{aggr}$ is an approximating operator of $T_{P, \mathcal{D}}^{aggr}$.

Proof

Follows from Proposition 7.1 and Proposition 7.2. \square

Definition 7.6

Given a three-valued structure $\tilde{\mathcal{D}}$, the $\tilde{\mathcal{D}}$ -Kripke-Kleene model, the $\tilde{\mathcal{D}}$ -well-founded model and the set of $\tilde{\mathcal{D}}$ -stable models of an aggregate program P are defined as the Kripke-Kleene, well-founded and the set of exact stable fixpoints of the $\Phi_{P, \tilde{\mathcal{D}}}^{agg}$ operator.

For logic programs without aggregates each $\Phi_{P, \tilde{\mathcal{D}}}^{agg}$ operator coincides with the operator Φ_P defined by Fitting (1985). So, the well-founded and stable semantics of aggregate programs is an extension of the well-founded and stable semantics of normal logic programs.

Notice that the semantics of an aggregate program P depends on $\tilde{\mathcal{D}}$ and, in particular, on the choice of the three-valued aggregates. This means that we still have a family of different semantics. This family can be ordered by precision. Not surprisingly, using more precise three-valued aggregates leads to more precise semantics.

The following proposition is a straightforward consequence of Proposition 7.1.

Proposition 7.4

For every pair of three-valued Σ -structures $\tilde{\mathcal{D}}_1$ and $\tilde{\mathcal{D}}_2$ and for every three-valued interpretation \tilde{I} , if $\tilde{\mathcal{D}}_1 \leq_p \tilde{\mathcal{D}}_2$ then $\Phi_{P, \tilde{\mathcal{D}}_1}^{agg}(\tilde{I}) \leq_p \Phi_{P, \tilde{\mathcal{D}}_2}(\tilde{I})$.

This proposition shows that $\Phi_{P, \tilde{\mathcal{D}}_1}^{agg}$ is a less precise approximation of $T_{P, \tilde{\mathcal{D}}}^{agg}$ than $\Phi_{P, \tilde{\mathcal{D}}_2}^{agg}$. So, by Theorem 2.2 we obtain the following result.

Theorem 7.5

Let P be an aggregate program and $\tilde{\mathcal{D}}_1$ and $\tilde{\mathcal{D}}_2$ be two three-valued Σ -structures such that $\tilde{\mathcal{D}}_1 \leq_p \tilde{\mathcal{D}}_2$. Then:

- the $\tilde{\mathcal{D}}_1$ -Kripke-Kleene model of P is less precise (in the \leq_p order) than the $\tilde{\mathcal{D}}_2$ -Kripke-Kleene model of P ;
- the $\tilde{\mathcal{D}}_1$ -well-founded model of P is less precise (in the \leq_p order) than the $\tilde{\mathcal{D}}_2$ -well-founded model of P ;
- every $\tilde{\mathcal{D}}_1$ -stable model is a $\tilde{\mathcal{D}}_2$ -stable model.

The semantics that we have defined in this section do not satisfy all the strong declarative properties of the ultimate semantics defined in the previous section. For example, the $\tilde{\mathcal{D}}$ -well-founded model of an aggregate program with monotone immediate consequence operator is not necessarily its least fixpoint. E.g. the program $\{p \leftarrow p \vee \neg p\}$ has a constant, hence monotone T_P with least fixpoint $\{p\}$, but in its well-founded model p is unknown. Also, substituting a formula for an equivalent formula in a rule body is not in general equivalence preserving. E.g. substituting *true* for $p \vee \neg p$ in the above program does not preserve equivalence. However, some interesting properties still hold.

Proposition 7.6

Let P and P' be aggregate programs such that P' is obtained by substituting an aggregate formula φ' for an aggregate formula φ in the body of a rule of P . If $\forall(\varphi \equiv \varphi')$ is satisfied in all three-valued $\Sigma(\Pi)$ -extensions of $\tilde{\mathcal{D}}$, then P and P' are equivalent under the $\tilde{\mathcal{D}}$ -Kripke-Kleene, $\tilde{\mathcal{D}}$ -well-founded and $\tilde{\mathcal{D}}$ -stable semantics.

Proof

Follows from the fact that P and P' have the same three-valued immediate consequence operators. \square

The three-valued equivalence condition in this proposition is strictly stronger than the two-valued equivalence condition in Proposition 6.2. For example $true$ and $p \vee \neg p$ are equivalent in two-valued semantics but not in three-valued.

Another important property is that in case of a stratified aggregate program P , the $\tilde{\mathcal{D}}$ -well-founded semantics and $\tilde{\mathcal{D}}$ -stable semantics coincide with the standard semantics as defined in Section 5.2, and it does not matter how the aggregate relations are approximated by $\tilde{\mathcal{D}}$.

Theorem 7.7

Let P be a stratified $\Sigma(\Pi)$ -aggregate program. For any three-valued Σ -structure $\tilde{\mathcal{D}}$ approximating \mathcal{D} , the $\tilde{\mathcal{D}}$ -well-founded model is two-valued and is equal to the standard model of P extending \mathcal{D} and to the unique $\tilde{\mathcal{D}}$ -stable model of P .

The proof of this result depends on the following lemma.

Lemma 7.8

Let φ be a closed $\Sigma(\Pi)$ -aggregate formula such that predicates of Π do not occur in an aggregate atom. Let \mathcal{D} be a Σ -structure and $\tilde{\mathcal{D}}$ be a three-valued Σ -structure which is two-valued on all predicates in Σ and approximates \mathcal{D} . Then for any three-valued interpretation (I_1, I_2) , if the predicates in Π occur only positively in φ then $\mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\varphi) = (\mathcal{H}_{\mathcal{D}(I_1)}(\varphi), \mathcal{H}_{\mathcal{D}(I_2)}(\varphi))$ and if the predicates in Π occur only negatively in φ then $\mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\varphi) = (\mathcal{H}_{\mathcal{D}(I_2)}(\varphi), \mathcal{H}_{\mathcal{D}(I_1)}(\varphi))$.

Proof

By simultaneous induction on the structure of φ . We give only the case when the predicates of Π occur only positively in φ . The proof of the other case is symmetric.

- For a pre-defined atom $p(t_1, \dots, t_n)$:

$$\begin{aligned} \mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(p(t_1, \dots, t_n)) &= \mathcal{H}_{\tilde{\mathcal{D}}}(p(t_1, \dots, t_n)) \\ &= (\mathcal{H}_{\mathcal{D}}(p(t_1, \dots, t_n)), \mathcal{H}_{\mathcal{D}}(p(t_1, \dots, t_n))). \end{aligned}$$

- For a user defined atom $p(t_1, \dots, t_n)$:

$$\begin{aligned} \mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(p(t_1, \dots, t_n)) &= (I_1, I_2)(p(\llbracket t_1 \rrbracket_{\mathcal{D}}, \dots, \llbracket t_n \rrbracket_{\mathcal{D}})) \\ &= (I_1(p(\llbracket t_1 \rrbracket_{\mathcal{D}}, \dots, \llbracket t_n \rrbracket_{\mathcal{D}})), I_2(p(\llbracket t_1 \rrbracket_{\mathcal{D}}, \dots, \llbracket t_n \rrbracket_{\mathcal{D}}))). \end{aligned}$$

- For a formula with negation $\neg\varphi$:

$$\begin{aligned} \mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\neg\varphi) &= \neg\mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\varphi) && \text{by Definition 7.4} \\ &= \neg(\mathcal{H}_{\mathcal{D}(I_2)}(\varphi), \mathcal{H}_{\mathcal{D}(I_1)}(\varphi)) && \text{by the induction hypothesis} \\ &= (\neg\mathcal{H}_{\mathcal{D}(I_1)}(\varphi), \neg\mathcal{H}_{\mathcal{D}(I_2)}(\varphi)) && \text{by definition of } \neg \\ &= (\mathcal{H}_{\mathcal{D}(I_1)}(\neg\varphi), \mathcal{H}_{\mathcal{D}(I_2)}(\neg\varphi)) && \text{by Definition 4.6.} \end{aligned}$$

- For a conjunction $\varphi \wedge \psi$:

$$\begin{aligned}
\mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\varphi \wedge \psi) &= \mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\varphi) \wedge \mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\psi) \\
&= (\mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}^1(\varphi) \wedge \mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}^1(\psi), \mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}^2(\varphi) \wedge \mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}^2(\psi)) \\
&= (\mathcal{H}_{\mathcal{D}(I_1)}(\varphi) \wedge \mathcal{H}_{\mathcal{D}(I_1)}(\psi), \mathcal{H}_{\mathcal{D}(I_2)}(\varphi) \wedge \mathcal{H}_{\mathcal{D}(I_2)}(\psi)) \\
&= (\mathcal{H}_{\mathcal{D}(I_1)}(\varphi \wedge \psi), \mathcal{H}_{\mathcal{D}(I_2)}(\varphi \wedge \psi)).
\end{aligned}$$

- The proofs for formulas of the form $\varphi \vee \psi$, $\exists x\varphi$, and $\forall x\varphi$ are analogous.
- Let $\mathbf{R}(\{\bar{x} \mid \varphi\}, t)$ be an aggregate atom. Since φ contains only pre-defined predicate symbols from Σ then

$$\mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\mathbf{R}(\{\bar{x} \mid \varphi\}, t)) = \mathcal{H}_{\tilde{\mathcal{D}}}(\mathbf{R}(\{\bar{x} \mid \varphi\}, t)).$$

Moreover, since the interpretation by $\tilde{\mathcal{D}}$ of all predicate symbols is two-valued we have (by Proposition 7.2):

$$\mathcal{H}_{\tilde{\mathcal{D}}}(\mathbf{R}(\{\bar{x} \mid \varphi\}, t)) = (\mathcal{H}_{\mathcal{D}}(\mathbf{R}(\{\bar{x} \mid \varphi\}, t)), \mathcal{H}_{\mathcal{D}}(\mathbf{R}(\{\bar{x} \mid \varphi\}, t))).$$

□

Proof of Theorem 7.7 (Sketch)

Given an interpretation $I \in \mathcal{I}$, let us define $I|_i$ as the restriction of I to the predicates of Π_i , and $I|_{\leq i}$ and $I|_{< i}$ as the restriction of I to the predicates of $\bigcup_{j \leq i} \Pi_j$, respectively those of $\bigcup_{j < i} \Pi_j$. We extend these notations also to three-valued interpretations. It is easy to see that for every $i = 1, \dots, \|P\|$ and every $\tilde{I}, \tilde{J} \in \mathcal{I}^c$, if $\tilde{I}|_{\leq i} = \tilde{J}|_{\leq i}$, then

$$\Phi_{P, \tilde{\mathcal{D}}}^{aggr}(\tilde{I})|_{\leq i} = \Phi_{P, \tilde{\mathcal{D}}}^{aggr}(\tilde{J})|_{\leq i}.$$

Moreover, for every $\tilde{I} \in \mathcal{I}^c$:

$$\Phi_{P, \tilde{\mathcal{D}}}^{aggr}(\tilde{I})|_i = \Phi_{P_i, \tilde{\mathcal{D}}(\tilde{I}|_{< i})}^{aggr}(\tilde{I}|_i).$$

It follows from Corollary 3.12 in (Vennekens et al. 2005) that \tilde{I} is the well-founded fixpoint of $\Phi_{P, \tilde{\mathcal{D}}}^{aggr}$ if and only if for every $i = 1, \dots, \|P\|$, $\tilde{I}|_i$ is the well-founded fixpoint of $\Phi_{P_i, \tilde{\mathcal{D}}(\tilde{I}|_{< i})}^{aggr}$.

The next step is to prove by induction that for every $i = 1, \dots, \|P\|$, the well-founded fixpoint $\tilde{I}|_i$ of $\Phi_{P_i, \tilde{\mathcal{D}}(\tilde{I}|_{< i})}^{aggr}$ is equal to $(I|_i, I|_i)$ where I is the standard model of P . This will show that the $\tilde{\mathcal{D}}$ -well-founded model \tilde{I} of P is equal to the standard model I of P . Fix i and assume that $\tilde{I}|_{< i} = (I|_{< i}, I|_{< i})$. Let $\Sigma_{i-1} = \Sigma \cup \bigcup_{j < i} \Pi_j$, $\mathcal{D}' = \mathcal{D}(I|_{< i})$ and $\tilde{\mathcal{D}}' = \tilde{\mathcal{D}}(\tilde{I}|_{< i})$. Notice that $\tilde{\mathcal{D}}'$ approximates \mathcal{D}' and $\tilde{\mathcal{D}}'$ is two-valued on all predicates of Σ_{i-1} (because $\tilde{I}|_{< i}$ is a two-valued interpretation). Using Lemma 7.8 we can show that for any three-valued Π_i -interpretation (I_1, I_2) ,

$$\Phi_{P_i, \tilde{\mathcal{D}}'}^{aggr}(I_1, I_2) = (T_{P_i, \mathcal{D}'}^{aggr}(I^1), T_{P_i, \mathcal{D}'}^{aggr}(I^2)).$$

By Theorem 2.4 follows that $\Phi_{P_i, \tilde{\mathcal{D}}'}^{aggr}$ is also the ultimate approximation of $T_{P_i, \mathcal{D}'}^{aggr}$. So, the $\tilde{\mathcal{D}}'$ -well-founded model of P_i is equal to the ultimate well-founded model of P_i extending \mathcal{D}' and, by Corollary 6.1, to the least fixpoint of $T_{P_i, \mathcal{D}'}^{aggr}$. □

Since the $\Phi_{P,\mathcal{D}}^{agg}$ operators are less precise than the ultimate approximation $U_{P,\mathcal{D}}^{agg}$ it follows by Theorem 2.2 that for stratified programs the ultimate well-founded model is also two-valued and is equal to the unique ultimate stable model and to the standard model of P .

In the next sections we define several different three-valued aggregate relations and study the semantics obtained from the corresponding $\Phi_{P,\mathcal{D}}^{agg}$ operator.

7.1 Trivial Approximating Aggregates

As a first example of a three-valued aggregate relation, we consider the least precise approximation of an aggregate.

Definition 7.7 (Trivial Approximating Aggregate)

Let $R \subseteq \wp(D_1) \times D_2$ be an aggregate relation. The *trivial approximating aggregate* $triv(R): \wp(D_1)^c \times D_2 \rightarrow \mathcal{THRE}$ of R is defined as follows:

$$triv(R)((S_1, S_2), d) = \begin{cases} (\mathbf{f}, \mathbf{t}) & \text{if } S_1 \neq S_2 \\ (R(S_1, d), R(S_1, d)) & \text{if } S_1 = S_2 \end{cases}$$

Proposition 7.9

Let R be an aggregate relation. For every three-valued aggregate relation \mathcal{R} of R , $triv(R) \leq_p \mathcal{R}$.

For a structure \mathcal{D} we define $triv(\mathcal{D})$ as the three-valued structure in which every aggregate relation R is interpreted with $triv(R)$. When \mathcal{D} is clear from the context we simply use $triv$.

The trivial approximating aggregates are very imprecise. Nevertheless, by Theorem 7.7, they suffice to model the semantics of the important class of stratified aggregate programs.

Corollary 7.10

Let P be a stratified aggregate program. The *triv*-well-founded model of P is two-valued and is equal to the standard model of P and the unique *triv*-stable model of P .

This corollary shows that even the weakest instance of the well-founded and stable semantics suffices to define the standard model approach for stratified aggregate programs.

7.2 Ultimate Approximating Aggregate

In this section we investigate the instance of the Φ_P^{agg} operator in which aggregate symbols are interpreted with the most precise three-valued aggregate relation, called the *ultimate approximating aggregate*. This three-valued aggregate relation is defined for all aggregate relations in a uniform way using a construction similar to that of ultimate approximations.

Definition 7.8 (Ultimate Approximating Aggregate)

Let $R \subseteq \wp(D_1) \times D_2$ be an aggregate relation. The *ultimate approximating aggregate* of R is a three-valued aggregate relation $ult(R): \wp(D_1)^c \times D_2 \rightarrow \mathcal{THRE}$ defined as:

$$\begin{aligned} ult(R)^1((S_1, S_2), d) &= \mathbf{t} \text{ if and only if } \forall S \in [S_1, S_2]: (S, d) \in R \\ ult(R)^2((S_1, S_2), d) &= \mathbf{t} \text{ if and only if } \exists S \in [S_1, S_2]: (S, d) \in R \end{aligned}$$

Proposition 7.11

For every aggregate relation R , $ult(R)$ is a three-valued aggregate relation and $ult(R)$ approximates R .

For a structure \mathcal{D} we define $ult(\mathcal{D})$ as the three-valued structure in which every aggregate relation R is interpreted with $ult(R)$. When \mathcal{D} is clear from the context we simply use ult .

The ultimate approximating aggregate $ult(R)$ is the most precise in the \leq_p -order among all possible three-valued aggregate relations.

Proposition 7.12

Let R be an aggregate relation. For every three-valued aggregate relation \mathcal{R} which approximates R , $\mathcal{R} \leq_p ult(R)$.

So, the $\Phi_{P,ult}^{aggr}$ is the most precise operator in the family of $\Phi_{P,\mathcal{D}}^{aggr}$ operators and by Proposition 7.5, the *ult*-well-founded and the *ult*-stable semantics are the most precise semantics of aggregate programs in this family. However, these semantics are still weaker than the ultimate well-founded and ultimate stable semantics from Section 6.

Now we look at characterizations of the ultimate approximating aggregate of some common aggregate functions. Such characterizations are useful for several purposes. First of all, they can be used in an implementation of the semantics. Second, they can be used for complexity analysis of the semantics.

For monotone and anti-monotone aggregates the truth value can be computed directly on the boundary multisets.

Proposition 7.13

Let $R: \wp(D_1) \times D_2$ be a monotone aggregate relation. Then $((S_1, S_2), d) \in ult(R)^1$ if and only if $(S_1, d) \in R$ and $((S_1, S_2), d) \in ult(R)^2$ if and only if $(S_2, d) \in R$.

Proposition 7.14

Let $R: \wp(D_1) \times D_2$ be an anti-monotone aggregate relation. Then $((S_1, S_2), d) \in ult(R)^1$ if and only if $(S_2, d) \in R$ and $((S_1, S_2), d) \in ult(R)^2$ if and only if $(S_1, d) \in R$.

Next, we look at extrema aggregates, possibly defined on infinite sets.

Proposition 7.15

The characterizations on Table 2 are correct.

Next, look at the aggregate functions CARD, SUM and PROD defined on finite sets.

R	$\ ((S_1, S_2), d) \in \text{ult}(\mathbb{R})^1 \text{ iff } \mid ((S_1, S_2), d) \in \text{ult}(\mathbb{R})^2 \text{ iff}$
MIN	$\left\ \begin{array}{l} d \in S_1 \wedge \text{MIN}(S_2, d) \\ d \in S_1 \wedge \text{MAX}(S_2, d) \end{array} \right\ \left\ \begin{array}{l} d \in S_2 \wedge \neg \exists d' \in S_1: d' < d \\ d \in S_2 \wedge \neg \exists d' \in S_1: d' > d \end{array} \right\ $
GLB	$\left\ \begin{array}{l} \text{GLB}(S_1, d) \wedge \text{GLB}(S_2, d) \\ \text{LUB}(S_1, d) \wedge \text{LUB}(S_2, d) \end{array} \right\ \left\ \begin{array}{l} \text{GLB}(S_1 \cup (S_2 \cap [d, \top]), d) \\ \text{LUB}(S_1 \cup (S_2 \cap [\perp, d]), d) \end{array} \right\ $

Table 2. Characterization of ultimate approximating aggregates of extrema aggregates.

Proposition 7.16 ($\text{ult}(\text{CARD})$)

$$\begin{aligned} ((S_1, S_2), d) \in \text{ult}(\text{CARD})^1 & \text{ if and only if } |S_1| = d = |S_2| \\ ((S_1, S_2), d) \in \text{ult}(\text{CARD})^2 & \text{ if and only if } |S_1| \leq d \leq |S_2| \end{aligned}$$

Proposition 7.17 ($\text{ult}(\text{SUM})^1$ and $\text{ult}(\text{PROD})^1$)

$$\begin{aligned} ((S_1, S_2), d) \in \text{ult}(\text{SUM})^1 & \text{ iff } \text{SUM}(S_1, d) \wedge \forall \bar{x} \in S_2 \setminus S_1: x_1 = 0 \\ ((S_1, S_2), d) \in \text{ult}(\text{SUM})^2 & \text{ iff } \exists S' \subseteq S_2 \setminus S_1: \text{SUM}(S_1 \cup S', d) \end{aligned}$$

$$\begin{aligned} ((S_1, S_2), d) \in \text{ult}(\text{PROD})^1 & \text{ iff } \text{PROD}(S_1, d) \wedge (0 \in S_1 \vee \forall \bar{x} \in S_2 \setminus S_1: x_1 = 1) \\ ((S_1, S_2), d) \in \text{ult}(\text{PROD})^2 & \text{ iff } \exists S' \subseteq S_2 \setminus S_1: \text{PROD}(S_1 \cup S', d) \end{aligned}$$

Note that the definition of $\text{ult}(\text{SUM})^2$ and $\text{ult}(\text{PROD})^2$ is simply a reformulation of the original definition of ult . In fact, Pelov () shows that the complexity of computing $\text{ult}(\text{SUM})^2$ and $\text{ult}(\text{PROD})^2$ is NP-complete so it is unlikely that any efficient algorithms can be found.

Finally, we look at combined aggregate relations of the form F_{\geq} and F_{\leq} where $F: \wp_F(D_1) \rightarrow D_2$ is an aggregate function on finite sets and \leq is a total order on D_2 . For all three aggregate functions CARD , SUM , and PROD we can give efficient algorithms for $\text{ult}(F_{\geq})^1$ and $\text{ult}(F_{\geq})^2$. We start with the following general result. Let $\min_F, \max_F: \wp_F(D_1)^c \rightarrow \wp_F(D_1)$ be functions which for a given finite three-valued set (S_1, S_2) return a set $S \in [S_1, S_2]$ such that $F(S)$ is minimal (resp. maximal) over the set $[S_1, S_2]$, i.e.

$$\begin{aligned} \forall S' \in [S_1, S_2]: F(\min_F(S_1, S_2)) & \leq F(S') \\ \forall S' \in [S_1, S_2]: F(\max_F(S_1, S_2)) & \geq F(S'). \end{aligned}$$

Note that for a given aggregate function F and a three-valued set (S_1, S_2) there may be more than one set in the interval $[S_1, S_2]$ with a minimal value of F . For example $\min_{\text{SUM}}(\emptyset, \{0\})$ can return either \emptyset or $\{0\}$.

The values of $\text{ult}(F_{\leq})$ and $\text{ult}(F_{\geq})$ can be computed using the \min_F and \max_F functions in the following way.

Proposition 7.18

Let $F: \wp_F(D_1) \rightarrow D_2$ be an aggregate function and \leq be a total order on D_2 . Then:

$$\begin{aligned} ((S_1, S_2), d) \in \text{ult}(F_{\geq})^1 & \text{ if and only if } F(\min_F(S_1, S_2)) \geq d \\ ((S_1, S_2), d) \in \text{ult}(F_{\geq})^2 & \text{ if and only if } F(\max_F(S_1, S_2)) \geq d \end{aligned}$$

and similarly for $\text{ult}(F_{\leq})$:

$$\begin{aligned} ((S_1, S_2), d) \in \text{ult}(F_{\leq})^1 & \text{ if and only if } F(\max_F(S_1, S_2)) \leq d \\ ((S_1, S_2), d) \in \text{ult}(F_{\leq})^2 & \text{ if and only if } F(\min_F(S_1, S_2)) \leq d. \end{aligned}$$

Proof

First note that, since $\{F(S) \mid S \in [S_1, S_2]\}$ is a finite totally ordered set, it always has a minimal and a maximal element. We give the proof for $\text{ult}(F_{\geq})^1$:

$$\begin{aligned} & ((S_1, S_2), d) \in \text{ult}(F_{\geq})^1 \\ \Leftrightarrow & \forall S \in [S_1, S_2]: (S, d) \in F_{\geq} \\ \Leftrightarrow & \forall S \in [S_1, S_2]: F(S) \geq d \\ \Leftrightarrow & \forall x \in \{F(S) \mid S \in [S_1, S_2]\}: x \geq d \\ \Leftrightarrow & F(\min_F(S_1, S_2)) \geq d. \end{aligned}$$

The proofs of the other cases are analogous. \square

So, to decide the first and second components of $\text{ult}(F_{\geq})((S_1, S_2), d)$ we need to compute the values $\min_F(S_1, S_2)$ and $\max_F(S_1, S_2)$. First, we show how to compute these two functions for any monotone or anti-monotone aggregate function.

Proposition 7.19

If F is a monotone aggregate function with respect to \leq then $\min_F(S_1, S_2) = S_1$ and $\max_F(S_1, S_2) = S_2$. If F is an anti-monotone aggregate function with respect to \leq then $\min_F(S_1, S_2) = S_2$ and $\max_F(S_1, S_2) = S_1$.

This proposition can be applied to all aggregate functions listed on Table 1.

For aggregate functions which are non-monotone the idea is to partition the under and over-estimate of the input three-valued set to subsets on which the aggregate function is monotone or anti-monotone. Then we combine the sets on which the function is monotone to obtain \min_F and the sets on which it is anti-monotone to obtain \max_F . We illustrate this idea for the SUM and PROD aggregate functions.

Below, S^+ denotes the set $\{(x_1, \dots, x_n) \in S \mid x_1 \geq 0\}$ and S^- denotes the set $\{(x_1, \dots, x_n) \in S \mid x_1 < 0\}$.

Proposition 7.20

$$\begin{aligned} \min_{\text{SUM}}(S_1, S_2) &= S_1^+ \cup S_2^- \\ \max_{\text{SUM}}(S_1, S_2) &= S_1^- \cup S_2^+. \end{aligned}$$

Proof

Clearly the set $S' \in [S_1, S_2]$ with minimal sum is obtained by taking S_1 and all tuples with negative numbers from $S_2 \setminus S_1$, that is

$$\min_{\text{SUM}}(S_1, S_2) = S_1 \cup (S_2 \setminus S_1)^- = S_1^+ \cup S_1^- \cup (S_2^- \setminus S_1^-) = S_1^+ \cup S_2^-.$$

□

The aggregate function PROD is non-monotone for sets with arbitrary real numbers as first argument but is monotone for sets with the first argument in the interval $[1, \infty)$ and anti-monotone for sets with the first argument in the interval $[0, 1]$. So, for PROD on non-negative real numbers we can give a similar algorithm as for SUM in Proposition 7.20. Below, $S^{[1, \infty)}$ denotes the set $\{(x_1, \dots, x_n) \in S \mid x_1 \in [1, \infty)\}$ and $S^{[0, 1]}$ denotes the set $\{(x_1, \dots, x_n) \in S \mid x_1 \in [0, 1]\}$.

Proposition 7.21

For the aggregate function $\text{PROD}^{\mathbb{R}^+} : \wp_F(\mathbb{R}^+) \rightarrow \mathbb{R}^+$,

$$\begin{aligned} \min_{\text{PROD}^{\mathbb{R}^+}}(S_1, S_2) &= S_1^{[1, \infty)} \cup S_2^{[0, 1]} \\ \max_{\text{PROD}^{\mathbb{R}^+}}(S_1, S_2) &= S_1^{[0, 1]} \cup S_2^{[1, \infty)}. \end{aligned}$$

The algorithms for PROD on the entire set of real numbers are more complicated and can be found in (Pelov).

As an application of the *ult*-well-founded semantics we consider a recursive formulation of the shortest path problem.

Example 7.1 (Shortest Path)

Consider the following formulation of the problem of finding the shortest path (Van Gelder 1992, Example 4.1):

$$sp(x, y, w) \leftarrow \text{MIN}(\{c \mid cp(x, y, c)\}, w).$$

$$cp(x, y, c) \leftarrow \text{edge}(x, y, c).$$

$$cp(x, y, c_1 + c_2) \leftarrow sp(x, z, c_1) \wedge \text{edge}(z, y, c_2).$$

The only difference between this program and the formulation of the shortest path in Example 5.4 is that we have replaced the *cp* predicate in the body of the second clause of *cp* with the *sp* predicate. We have incorporated the knowledge that any shortest path of length $n + 1$ must be an extension of a shortest path of length n . This fact is the basis of Dijkstra's algorithm. However, the program is no longer stratified because the *sp/3* predicate depends on itself through the MIN aggregate relation which is non-monotone. □

It turns out the above program is only correct under certain conditions on the graph.

Proposition 7.22

Let $\text{edge}^{\mathcal{D}}$ be a graph with the property that for any pair of nodes a and b , if there is a path from a to b , there is a shortest path from a to b . In the *ult*-well-founded model of the shortest path program from Example 7.1 an atom $sp(a, b, w)$ is:

- *true* if a shortest path between a and b exists and has weight w ;
- *false* otherwise.

The proof of the proposition and a deeper analysis of this program is given in Appendix A. There are several types of graphs for which we can show that when there is a path from one node to another, there is a shortest path between these nodes: e.g. acyclic finite graphs, finite graphs with non-negative weights and infinite graphs with weights in \mathbb{N}_0 . It follows from Proposition 7.22 that for these types of graphs, the programs in Example 5.4 and Example 7.1 are equivalent.

There are also types of graphs which do not satisfy the condition. Connected nodes without shortest path can occur if there is a cycle with a negative weight between the two nodes. It can also occur in infinite graphs. An example of such a graph is $\{(0, n + 2, 1), (n + 2, 1, 1/n + 2) \mid n \in \mathbb{N}\}$; although there are an infinite number of paths from 0 to 1, there is no shortest path between 0 and 1 (see Ross and Sagiv (1997) for another example). In the appendix, we show that in such graphs, the well-founded model may be three-valued or may even contain erroneous $sp(a, b, w)$ atoms, i.e. there are paths between a and b with strictly less weight than w .

We conclude the section on ultimate approximating aggregates by showing that for definite aggregate programs, the *ult*-well-founded and *ult*-stable semantics are equal to the least fixpoint of the $T_{P, \mathcal{D}}^{agg}$ which we defined in Section 5.1. The key to the proof of this result is the following lemma.

Lemma 7.23

Let $\tilde{\mathcal{D}}$ be a three-valued structure approximating \mathcal{D} and let (I_1, I_2) be a three-valued interpretation. If φ is a closed positive aggregate formula then $\mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\varphi) = (\mathcal{H}_{\mathcal{D}(I_1)}(\varphi), \mathcal{H}_{\mathcal{D}(I_2)}(\varphi))$. If φ is a closed negative aggregate formula then $\mathcal{H}_{\tilde{\mathcal{D}}(I_1, I_2)}(\varphi) = (\mathcal{H}_{\mathcal{D}(I_2)}(\varphi), \mathcal{H}_{\mathcal{D}(I_1)}(\varphi))$.

Proof

The proof extends that of Lemma 7.8 with several new cases when φ is an aggregate atom $R(\{\bar{x} \mid \psi\}, t)$, ψ contains defined predicates and $R^{\mathcal{D}}$ is either a monotone or an anti-monotone aggregate relation. We consider only the case when $R^{\mathcal{D}}$ is a monotone aggregate relation and ψ is a positive aggregate formula. The other three cases ($R^{\mathcal{D}}$ monotone and ψ negative and $R^{\mathcal{D}}$ anti-monotone and ψ positive or negative) are symmetric. First, note that since $\psi(\bar{x})$ is a positive formula then for every tuple of domain elements \bar{d} , $\psi(\bar{d})$ is also positive. So,

$$\mathcal{H}_{(I_1, I_2)}(\psi(\bar{d})) = (\mathcal{H}_{I_1}(\psi(\bar{d})), \mathcal{H}_{I_2}(\psi(\bar{d})))$$

and consequently

$$\llbracket \{\bar{x} \mid \psi\} \rrbracket_{\tilde{\mathcal{D}}(I_1, I_2)} = (\llbracket \{\bar{x} \mid \psi\} \rrbracket_{\mathcal{D}(I_1)}, \llbracket \{\bar{x} \mid \psi\} \rrbracket_{\mathcal{D}(I_2)}).$$

We have:

$$\begin{aligned}
\mathcal{H}_{\bar{\mathcal{D}}(I_1, I_2)}(\mathbb{R}(\{\bar{x} \mid \psi\}, t)) &= \text{ult}(\mathbb{R})(\llbracket \{\bar{x} \mid \psi\} \rrbracket_{\bar{\mathcal{D}}(I_1, I_2)}, \llbracket t \rrbracket) \\
&= \text{ult}(\mathbb{R})(\llbracket \{\bar{x} \mid \psi\} \rrbracket_{\mathcal{D}(I_1)}, \llbracket \{\bar{x} \mid \psi\} \rrbracket_{\mathcal{D}(I_2)}, \llbracket t \rrbracket) \\
\text{(by Proposition 7.13)} &= (\mathbb{R}(\llbracket \{\bar{x} \mid \psi\} \rrbracket_{\mathcal{D}(I_1)}, \llbracket t \rrbracket), \mathbb{R}(\llbracket \{\bar{x} \mid \psi\} \rrbracket_{\mathcal{D}(I_2)}, \llbracket t \rrbracket)) \\
&= (\mathcal{H}_{\mathcal{D}(I_1)}(\mathbb{R}(\{\bar{x} \mid \psi\}, t)), \mathcal{H}_{\mathcal{D}(I_2)}(\mathbb{R}(\{\bar{x} \mid \psi\}, t))).
\end{aligned}$$

□

Theorem 7.24

Let P be a definite aggregate program. Then P has a two-valued *ult*-well-founded model (M, M) which is also the single *ult*-stable model. Moreover $M = \text{lfp}(T_{P, \mathcal{D}}^{\text{aggr}})$.

Proof

From Lemma 7.23 follows that if P is a positive aggregate program then

$$\Phi_{P, \text{ult}}^{\text{aggr}}(I_1, I_2) = (T_{P, \mathcal{D}}^{\text{aggr}}(I_1), T_{P, \mathcal{D}}^{\text{aggr}}(I_2)).$$

By Theorem 2.4 follows that $\Phi_{P, \text{ult}}^{\text{aggr}}$ is also the ultimate approximation of $T_{P, \mathcal{D}}^{\text{aggr}}$. So, the *ult*-well-founded model of P is equal to the ultimate well-founded model of P extending \mathcal{D} and, by Corollary 6.1, to the least fixpoint of $T_{P, \mathcal{D}}^{\text{aggr}}$. □

7.3 Bound Approximating Aggregate

The ultimate approximating aggregates have the disadvantage that they may have a higher complexity than the aggregate relations which they approximate. We already mentioned that the complexity of $\text{ult}(\text{SUM})^2$ and $\text{ult}(\text{PROD})^2$ is NP-complete (Pelov) while the complexity of SUM and PROD is polynomial. In this section we define a less precise approximating operator for aggregate functions that are defined on totally ordered finite sets of numbers.

Definition 7.9

Let $F: \wp_F(D_1) \rightarrow D_2$ be an aggregate function and $\langle D_2, \leq \rangle$ be a totally ordered set. Define the *three-valued aggregate relation* $\text{bnd}(F): \wp_F(D_1)^c \times D_2 \rightarrow \mathcal{THRE}$ as follows

$$\begin{aligned}
((S_1, S_2), d) &\in \text{bnd}(F)^1 \text{ if and only if } F(\min_F(S_1, S_2)) = d = F(\max_F(S_1, S_2)) \\
((S_1, S_2), d) &\in \text{bnd}(F)^2 \text{ if and only if } F(\min_F(S_1, S_2)) \leq d \leq F(\max_F(S_1, S_2)).
\end{aligned}$$

Note that by using Proposition 7.18, the definition of $\text{bnd}(F)$ is equivalent to $\text{ult}(F_{\geq}) \wedge \text{ult}(F_{\leq})$ where \wedge is the greatest lower bound in \mathcal{THRE} with respect to the \leq order (see Example 2.1).

Proposition 7.25

Let $F: \wp_F(D_1) \rightarrow D_2$ be an aggregate function and $\langle D_2, \leq \rangle$ be a totally ordered set. Then $\text{bnd}(F)$ is a three-valued aggregate relation of F .

The first component of $\text{bnd}(F)$ turns out to be equivalent to the ultimate approximating aggregate.

Proposition 7.26

Let $F: \wp_F(D_1) \rightarrow D_2$ be an aggregate function and \leq a total order on D_2 . Then $\text{bnd}(F)^1 = \text{ult}(F)^1$.

However, for most aggregate functions, $\text{bnd}(F)$ can be less precise than $\text{ult}(F)$ in the second component, i.e. $\text{ult}(F)^2 < \text{bnd}(F)^2$.

Example 7.2

The pair $(\{\emptyset, \{1, 3\}\}, 2)$ is not in the relation $\text{ult}(\text{SUM})^2$ because there is no set $S \in [\emptyset, \{1, 3\}]$ such that $\text{SUM}(S) = 2$. On the other hand $(\{\emptyset, \{1, 3\}\}, 2) \in \text{bnd}(\text{SUM})^2$ because

$$\begin{aligned} b_1 &= \text{SUM}(\min_{\text{SUM}}(\emptyset, \{1, 3\})) = \text{SUM}(\emptyset) = 0, \\ b_2 &= \text{SUM}(\max_{\text{SUM}}(\emptyset, \{1, 3\})) = \text{SUM}(\{1, 3\}) = 4, \end{aligned}$$

and $b_1 \leq 2 \leq b_2$. So, $\text{ult}(\text{SUM})^2 \subset \text{bnd}(\text{SUM})^2$ and consequently $\text{bnd}(\text{SUM}) <_p \text{ult}(\text{SUM})$.

7.4 On the Complexity

In this section we prove a simple complexity result. A full analysis of the complexity of model generation for aggregate programs is beyond the scope of this paper but we show that despite the second order nature of aggregates, model generation for aggregate programs may remain tractable under an appropriate choice of the three valued aggregates.

The type of computational problem considered here is the model extension problem (Mitchell and Ternovska 2005): given a signature $\Sigma(\Pi)$, an aggregate program P based on $\Sigma(\Pi)$ and an input Σ -structure \mathcal{D} which is two-valued on all predicates and three-valued on aggregate symbols, compute the Kripke-Kleene model, the well-founded model or an exact stable model. Informally, we are interested here in the complexity for instances of these problems with fixed P , fixed $\Sigma(\Pi)$, and “fixed” interpretation of the aggregate symbols in $\text{Aggr}(\Sigma)$ and for varying but finite input $\Sigma \setminus \text{Aggr}(\Sigma)$ -structures \mathcal{D}_o . The problem with this intuition is that the interpretation $\mathbb{R}^{\mathcal{D}}$ of a given aggregate symbol \mathbb{R} in this class is of course not really fixed: it varies with the input structure \mathcal{D}_o . We are interested in classes of problems where for example the sum predicate is systematically interpreted by its ultimate approximating aggregate, but evidently, the sum aggregate relation and its ultimate approximating aggregate depend on the domain of the input structure \mathcal{D}_o .

To circumvent this problem, we introduce the following concepts. Let us fix an aggregate program P based on signature $\Sigma(\Pi)$. Consider the class \mathcal{C} of two-valued structures of $\Sigma \setminus \text{Aggr}(\Sigma)$ with a finite domain (i.e. with finite domains for every sort s). We assume a given function X from \mathcal{C} to the class of Σ -structures such that for each $\mathcal{D}_o \in \mathcal{C}$, $X(\mathcal{D}_o)$ extends \mathcal{D}_o with three-valued aggregates \mathcal{R} for each aggregate symbol $\mathbb{R} \in \text{Aggr}(\Sigma)$. Moreover, we assume that for each aggregate symbol $\mathbb{R} \in \text{Aggr}(\Sigma)$, there are two Turing machines which, for (an appropriate encoding of) arbitrary $\mathcal{D}_o \in \mathcal{C}$ and arbitrary well-typed pair (S, d) consisting of a

three-valued set \tilde{S} and a domain element d , compute respectively $(\mathbb{R}^{X(\mathcal{D}_o)})^1(\tilde{S}, d)$ and $(\mathbb{R}^{X(\mathcal{D}_o)})^2(\tilde{S}, d)$.

Let $X(\mathcal{C})$ be the image class of \mathcal{C} under the function X . The main result of this section is that if each three-valued aggregate $\mathcal{R} \in \text{Aggr}(\Sigma)$ is polynomially computable in the size of the domain of the input structure, i.e., if for arbitrary \mathcal{D}_o , \tilde{S} and d , the Turing machines associated to \mathcal{R} can compute $(\mathbb{R}^{X(\mathcal{D}_o)})^1(\tilde{S}, d)$ and $(\mathbb{R}^{X(\mathcal{D}_o)})^2(\tilde{S}, d)$ in polynomial time in the size of the domain of \mathcal{D}_o (i.e. the total number of elements in all domains $s^{\mathcal{D}_o}$ of all sorts s), then the following holds:

Theorem 7.27

- computing the Kripke-Kleene model of a program P extending a structure in the class $X(\mathcal{C})$ is in P;
- computing the well-founded model of a program P extending a structure in the class $X(\mathcal{C})$ is in P;
- deciding the existence of an exact stable model of a program P extending a structure in the class $X(\mathcal{C})$ is in NP.

Note that, in the common case, computing the value of an aggregate is polynomial in the size of the input three-valued set \tilde{S} (of a fixed type $s_1 \times \dots \times s_n$). Then, since the number of elements in such a set is bounded by a polynomial in the size of the domain of \mathcal{D}_o , computing the value of the aggregate is certainly polynomial in the size of the domain of the input structure \mathcal{D}_o .

Proof

Let L be a (finite) lattice and A an approximation operator on L^c . Suppose that n is the length of the longest chain in L . The computation of the Kripke-Kleene and well-founded fixpoint of A and the test whether a lattice element is an exact stable fixpoint of A is done by monotone fixpoint computations. It is easy to see that the number of applications of A to compute its Kripke-Kleene fixpoint is bound by n . Also testing whether a lattice element is an exact stable fixpoint of A requires at most n applications of A . Because the computation of the well-founded fixpoint involves an embedded fixpoint computation, its computation takes at most n^2 applications of A .

Let us now consider a model extension problem for fixed $P, \Sigma(\Pi)$ and aggregate extension function X . Given an input $\Sigma \setminus \text{Aggr}(\Sigma)$ -structure \mathcal{D}_o , the lattice in which the computations take place is the power-set lattice $\mathcal{I} = \wp(\text{base}_{\mathcal{D}_o}(\Pi))$. The maximal chain length in this lattice is the number of facts, i.e. the cardinality of $\text{base}_{\mathcal{D}_o}(\Pi)$. This number is polynomial in the size of the domain of \mathcal{D}_o . It follows then that all we need to prove to obtain the desired complexity results is that for any given pair $\tilde{J} \in \mathcal{I}$, we can compute $\Phi_{P, X(\mathcal{D}_o)}^{\text{aggr}}(\tilde{J})$ in polynomial time in the size of the domain.

From Definition 7.5, it follows that $\Phi_{P, X(\mathcal{D}_o)}^{\text{aggr}}(\tilde{J})$ corresponds to computing the truth value of the bodies of all rule instances $A \leftarrow \varphi \in \text{inst}(P)$. It is clear that the number of rule instances is polynomial in the size of the domain of \mathcal{D}_o . Therefore, all we need to prove is that for an arbitrary aggregate formula $\varphi[\bar{x}]$ with free variables \bar{x} , the truth value of $\varphi[\bar{x}/\bar{d}]$ in \tilde{J} , for arbitrary $\mathcal{D}_o \in \mathcal{C}$ and tuple \bar{d} of domain elements, can be computed in polynomial time in the domain size of \mathcal{D}_o .

In case φ is a first order formula, the polynomial computability of its truth value with respect to a three-valued structure is proven by induction on the structure of φ . We need to extend this proof with the additional case that φ is an aggregate atom. Computing the truth value of an aggregate atom $R(s, d)$ requires firstly, to evaluate the contained set expression s and compute its three-valued set \tilde{S} , and secondly, to evaluate the truth value of $(\mathcal{R}^{X(\mathcal{D}_o)})^1(\tilde{S}, d)$ or $(\mathcal{R}^{X(\mathcal{D}_o)})^2(\tilde{S}, d)$. It follows straightforwardly from the induction hypothesis that the first task can be done in polynomial time in the domain size of \mathcal{D}_o , whereas the second task is polynomial by assumption. \square

Our main motivation for developing the semantic framework of this section was the high complexity of the ultimate well-founded and stable semantics as defined in Section 6. This result shows that under appropriate choice of the three-valued aggregates, we indeed obtain weaker but tractable Kripke-Kleene and well-founded semantics.

The above theorem is subject to a strong limitation, in particular the restriction to *finite* structures. Many of the interesting applications of aggregates (including the company control and the shortest path problem) contain integer or real numbered domains. Frequently used aggregates such as SUM and CARD range over these infinite domains. Clearly, in the context of infinite domains, only strong syntactical conditions on the form of rules can guarantee termination or tractability of the model generation process. But this is the case whether the program contains aggregate expressions or not. To discover such conditions is an important topic for future research but it is beyond the scope of this paper to investigate this issue.

8 Related Work

Aggregate relations are closely related to generalized quantifiers (Lindström 1966). An example of a generalized quantifier is $Most(A, B)$ which expresses that most elements of set A belong to set B . Clearly, this relation can be viewed as a binary aggregate relation with two set arguments. Standardly, the notion of generalized quantifier is formalized in a slightly more involved way than as a second order predicate in an arbitrary domain. Instead the concept is formalized as a class of structures closed under isomorphism. For example, $Most$ could be formalized as the class of all structures consisting of a domain D and a binary relation $M \subseteq \wp(D) \times \wp(D)$ consisting of all pairs (A, B) such that A is finite and more than half of the elements of A belong to B . In this way, a domain independent characterization of the generalized quantifier is obtained. Aggregates could be formalised similarly. For example, the cardinality aggregate could be formalised as the class of all structures consisting of a domain D and a binary relation $C \subset \wp(D) \times \mathbb{N}$ containing all tuples (S, n) such that S is a subset of D containing n elements. An extensive study of generalized quantifiers in three-valued logic is done by van Eijck (1996). The notion of a *supervaluation interpretation* of a generalized quantifier, as defined there, coincides with our notion of ultimate approximating aggregate of the corresponding aggregate relation.

In the context of logic programming, many different approaches to aggregates have been proposed. Below, we discuss a selection of them.

The class of *monotonic aggregate programs* (Mumick et al. 1990) is very similar to the class of definite aggregate programs. A monotonic program is a program in which every rule is monotonic. A monotonic rule is a rule r such that $I \models \text{body}(r)$ and $I \subseteq J$ implies $J \models \text{body}(r)$ for any pair of interpretations I and J . Although this is a semantic definition of monotonicity, the authors introduce a sufficient syntactic condition for monotonicity of a rule. Essentially, an aggregate atom can appear only in formulas of the form

$$\exists zR(\{\bar{x} \mid q(\bar{x})\}, z) \wedge p(z, t) \quad (1)$$

where p is a pre-defined relation. Moreover, the satisfiability of this formula must be monotone. In our syntax (1) can be expressed as the aggregate atom

$$R_P(\{\bar{x} \mid q(\bar{x})\}, t) \quad (2)$$

using the derived aggregate relation R_P . Then, the condition that the satisfiability of (1) is monotone is equivalent to the condition that R_P is a monotone aggregate relation (and consequently (2) is a positive aggregate atom). The notion of positive aggregate atoms is simpler and, in our opinion, more natural than the condition of monotonic literals of (Mumick et al. 1990).

In later work, aggregates have been introduced also in the context of general logic programs with negation under the stable semantics. Two similar approaches are found in (Kemp and Stuckey 1991; Gelfond 2002). Typical for these approaches is that aggregates are handled in a similar way as negation, in the sense that when computing the reduct of an aggregate program with respect to an interpretation I , negative literals `not` 1 and aggregate atoms are evaluated with respect to I . This leads to a semantics which accepts non-minimal models. Aggregates are also present in two of the main ASP-systems, namely `smodels` and `dlv`. The `smodels` system offers cardinality and sum aggregates in the form of *weight constraints* (Simons et al. 2002). For a more extensive discussion on the relationship with the proposals of (Kemp and Stuckey 1991; Gelfond 2002) and the stable semantics of weight constraints, we refer to (Pelov et al. 2004; Pelov).

Here, we focus on the extension of the stable semantics of disjunctive logic programs to programs with aggregates as proposed in (Faber et al. 2004). This approach is similar to ours, in the sense that it also aims to provide a semantics for (disjunctive) logic programs with negation and arbitrary aggregate relations, and it extends ours by allowing disjunctive rules. Below, we recall the main concepts of this approach.

Definition 8.1 (Faber et al. 2004)

The *reduct* P^I of an aggregate program P with respect to an interpretation I is a program obtained from $\text{inst}(P)$ by deleting all rules in which a literal or an aggregate atom in the body is false in I . An interpretation I is a *FLP-stable model* of P if I is a minimal model of P^I .

Note that this definition also works for rules with disjunction in the head. Also, this approach does not rely on three-valued aggregates.

Proposition 8.1

Let $\tilde{\mathcal{D}}$ be a three-valued structure. If I is a $\tilde{\mathcal{D}}$ -stable model of P then I is a FLP-stable model of P .

Proof

We will show that any $\tilde{\mathcal{D}}$ -stable model I is also a $\tilde{\mathcal{D}}$ -stable model of P^I . Because stable models are minimal models, this will imply that I is a FLP-stable model. We start by showing

$$\forall J \subseteq I: \Phi_{P^I, \tilde{\mathcal{D}}}^{aggr,1}(J, I) = \Phi_{P, \tilde{\mathcal{D}}}^{aggr,1}(J, I). \quad (*)$$

This will imply that $\text{lfp}(\Phi_{P^I, \tilde{\mathcal{D}}}^{aggr,1}(\cdot, I)) = \text{lfp}(\Phi_{P, \tilde{\mathcal{D}}}^{aggr,1}(\cdot, I)) = I$, so I will be a stable model of P^I . To show (*) we only need to look at the rules $r \in P - P^I$ and show that $\mathcal{H}_{\tilde{\mathcal{D}}(J, I)}^1(\text{body}(r)) = \mathbf{f}$. The definition of reduct implies that for such a rule $\mathcal{H}_{\mathcal{D}(I)}(\text{body}(r)) = \mathbf{f}$ where \mathcal{D} is the (two-valued) structure approximated by $\tilde{\mathcal{D}}$. We also have that $(J, I) \leq_p (I, I)$ which implies that

$$\mathcal{H}_{\tilde{\mathcal{D}}(J, I)}^1(\text{body}(r)) \leq \mathcal{H}_{\tilde{\mathcal{D}}(I, I)}^1(\text{body}(r)) = \mathcal{H}_{\mathcal{D}(I)}(\text{body}(r)) = \mathbf{f}$$

So, $\mathcal{H}_{\tilde{\mathcal{D}}(J, I)}^1(\text{body}(r)) = \mathbf{f}$. \square

The next example gives a program for which the two semantics disagree.

Example 8.1

Consider the following aggregate program P :

$$\begin{aligned} r &\leftarrow \text{CARD}_{\neq}(\{x \mid p(x)\}, 1). \\ p(A) &\leftarrow r. \\ p(B) &\leftarrow r. \\ p(A) &\leftarrow p(B). \\ p(B) &\leftarrow p(A). \end{aligned}$$

The program, interpreted as a set of aggregate formulas in which \leftarrow is a classical implication, has only one model $M = \{r, p(A), p(B)\}$. This is also a FLP-stable model because $P^M = P$ and M is also a minimal model of P . However, the program does not have an ultimate total stable model and, consequently, it does not have a total stable model for any less precise approximating operator.

To illuminate what is going on in this example, let us make the following observation. In the context of the Herbrand universe $\{A, B\}$, the aggregate atom $\text{CARD}_{\neq}(\{x \mid p(x)\}, 1)$ expresses that p has either zero or two elements. Or, in each two-valued Herbrand interpretation this atom is equivalent to

$$\neg p(A) \wedge \neg p(B) \vee p(A) \wedge p(B). \quad (3)$$

In fact, if we interpret CARD_{\neq} by the ultimate approximating aggregate, then a simple case analysis shows that, in each three-valued Herbrand interpretation, the truth values of the aggregate atom and of the formula (3) are identical². It follows

² The three-valued equivalence of $\text{ult}(\text{CARD}_{\neq})$ and (3) is an application of a general transformation of ultimate approximations of aggregate atoms to formulas (Pelov, Section 5.3.6).

that the program P and the program obtained by substituting the rules

$$\begin{aligned} r &\leftarrow \neg p(A) \wedge \neg p(B). \\ r &\leftarrow p(A) \wedge p(B). \end{aligned}$$

for the first rule in P , have identical three-valued immediate consequence operators and hence, have identical stable models³. The second program is a standard logic program and it is easy to see that it has no stable models. In particular, if we compute the reduct under $M = \{r, p(A), p(B)\}$, only the second new rule remains and together with the rest of the rules the least model is \emptyset . Thus M is not a stable model of the translated program. \square

The above example illustrates a natural principle of the semantics defined in our framework. Substituting an aggregate atom by an aggregate free formula which is equivalent with respect to two-valued semantics, preserves ultimate well-founded and ultimate stable models. Substituting an aggregate atom by an aggregate free formula which is equivalent with respect to three-valued semantics preserves the standard well-founded and stable semantics. As shown by the example, this principle is not satisfied by the semantics defined in (Faber et al. 2004).

Nevertheless, the class of programs for which the two semantics disagree seems to be rather small. It contains only programs with a recursion over a non-monotone aggregate. Characterizing formally this class is an interesting direction for future work, as it will allow to use the results, techniques, and algorithms which we developed for the semantics of Faber et al. (2004) and vice versa.

9 Conclusions and Future Work

In this paper we presented a systematic extension of all major semantics of logic programs to aggregate programs: the least fixpoint semantics (van Emden and Kowalski 1976), the standard model of stratified programs (Apt et al. 1988), the supported model semantics (Apt et al. 1988), the Kripke-Kleene semantics (Fitting 1985), the stable model semantics (Gelfond and Lifschitz 1988), the well-founded semantics (Van Gelder et al. 1991), and the ultimate stable and ultimate well-founded semantics (Denecker et al. 2004). The extension of the stable and well-founded semantics is not unique but is parameterized by how aggregate relations are extended to three-valued relations on three-valued sets. We studied three instances of these semantics. Two of them are the least precise (called *triv*) and the most precise (called *ult*) in this family and they are defined in a uniform way for all aggregate relations. The third instance (called *bnd*) is defined only for aggregate functions on totally ordered sets. For some aggregates, most notably SUM and PROD the *bnd*-semantics is strictly less precise than the *ult* semantics. Although, we did not present here a full complexity analysis, the advantage of the *bnd*-semantics over the *ult*-semantics is that it has a lower complexity (Pelov).

³ Stronger, both three-valued and two-valued immediate consequence operators coincide, and hence, all semantics of the two programs based on these operators coincide, including *ult*-Kripke-Kleene, *ult*-stable and *ult*-well-founded semantics.

We also showed that all important properties and relationships of the original semantics are preserved in the extension. For example, a stratified aggregate program P has a two-valued \tilde{D} -well-founded model which is equal to the unique \tilde{D} -stable model of P and the standard model of P for any three-valued interpretation \tilde{D} of the aggregate relations. A similar result for a definite aggregate program P is that the *ult*-well-founded model is equal to the unique *ult*-stable model and the least fixpoint model of P . Another important property of all stable semantics which we define is that stable models are always minimal.

References

- APT, K. R., BLAIR, H. A., AND WALKER, A. 1988. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, Chapter 2, 89–148.
- DELL'ARMI, T., FABER, W., IELPA, G., LEONE, N., AND PFEIFER, G. 2003. Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in DLV. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, G. Gottlob, Ed. Morgan Kaufmann, 847–852.
- DENECKER, M. 2000. Extending classical logic with inductive definitions. In *1st International Conference on Computational Logic*, J. Lloyd et al., Eds. Lecture Notes in Artificial Intelligence, vol. 1861. Springer, 703–717.
- DENECKER, M., MAREK, V., AND TRUSZCZYŃSKI, M. 2000. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In *Logic-based Artificial Intelligence*, J. Minker, Ed. Kluwer Academic Publishers, 127–144.
- DENECKER, M., MAREK, V., AND TRUSZCZYŃSKI, M. 2003. Uniform semantic treatment of default and autoepistemic logics. *Artificial Intelligence* 143, 1, 79–122.
- DENECKER, M., MAREK, V., AND TRUSZCZYŃSKY, M. 2004. Ultimate approximations and its application in nonmonotonic knowledge representation. *Information and Computation* 192, 1, 84–121.
- DENECKER, M., PELOV, N., AND BRUYNNOOGHE, M. 2001. Ultimate well-founded and stable model semantics for logic programs with aggregates. In *17th International Conference on Logic Programming*, P. Codognot, Ed. LNCS, vol. 2237. Springer, 212–226.
- FABER, W., LEONE, N., , AND PFEIFER, G. 2004. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of the 9th European Conference on Artificial Intelligence (JELIA)*. LNCS, vol. 3229. Springer, 200–212.
- FITTING, M. 1985. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming* 2, 4, 295–312.
- GELFOND, M. 2002. Representing knowledge in A-Prolog. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, A. C. Kakas and F. Sadri, Eds. Lecture Notes in Computer Science, vol. 2408. Springer, 413–451.
- GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Logic Programming, Proc. of the 5th International Conference and Symposium*, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, 1070–1080.
- KEMP, D. B. AND STUCKEY, P. J. 1991. Semantics of logic programs with aggregates. In *Proceedings of the International Logic Programming Symposium*, V. A. Saraswat and K. Ueda, Eds. MIT Press, 387–401.

- LINDSTRÖM, P. 1966. First order predicate logic with generalized quantifiers. *Theoria* 32, 186–195.
- MITCHELL, D. AND TERNOVSKA, E. 2005. A framework for representing and solving np search problems. In *AAAI-05*. 430–435.
- MUMICK, I. S., PIRAHESH, H., AND RAMAKRISHNAN, R. 1990. The magic of duplicates and aggregates. In *16th International Conference on Very Large Data Bases*, D. McLeod, R. Sacks-Davis, and H.-J. Schek, Eds. Morgan Kaufmann, 264–277.
- PELOV, N. Semantics of logic programs with aggregates. Ph.D. thesis, K.U.Leuven.
- PELOV, N., DENECKER, M., AND BRUYNOOGHE, M. 2004. Partial stable semantics for logic programs with aggregates. In *7th International Conference on Logic Programming and Nonmonotonic Reasoning*, V. Lifschitz and I. Niemelä, Eds. LNCS, vol. 2923. Springer, 207–219.
- ROSS, K. A. AND SAGIV, Y. 1997. Monotonic aggregation in deductive databases. *Journal of Computer and System Sciences* 54, 1, 79–97.
- SIMONS, P., NIEMELÄ, I., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138, 1–2, 181–234.
- VAN EIJCK, J. 1996. Quantifiers and partiality. In *Quantifiers, Logic, and Language*, J. van der Does and J. van Eijck, Eds. CSLI, 105–144.
- VAN EMDEN, M. H. AND KOWALSKI, R. A. 1976. The semantics of predicate logic as a programming language. *Journal of the ACM* 23, 4, 733–742.
- VAN GELDER, A. 1992. The well-founded semantics of aggregation. In *11th ACM Symposium on Principles of Database Systems*. ACM Press, 127–138.
- VAN GELDER, A., ROSS, K. A., AND SCHLIPF, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 38, 3, 620–650.
- VAN NUFFELEN, B. AND DENECKER, M. 2000. Problem solving in ID-logic with aggregates: some experiments. In *8th International Workshop on Nonmonotonic Reasoning, special track on Abductive Reasoning*, A. K. Mark Denecker, Ed. Breckenridge, Colorado, USA.
- VENNEKENS, J., GILIS, D., AND DENECKER, M. 2005. Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Transactions on Computational Logic*. accepted.

Appendix A Proof of the Shortest Path Theorem

Before proving Proposition 7.22, we first introduce some concepts.

We define the *length* of a path as its number of edges. We define a partial function $sp(.,.)$ as: $sp(a, b) = w$ if there is a shortest path from a to b and its weight is w . Note that it is possible that there is a path from a to b while there is not a shortest path. E.g., consider the graph with edges $\{(a, a, -1), (a, b, 1)\}$. It has no shortest paths.

Proposition 7.22

Let $edge^D$ be a graph with the property that for any pair of nodes a and b , if there is a path from a to b , there is a shortest path from a to b . In the *ult*-well-founded model of the shortest path program from Example 7.1 an atom $sp(a, b, w)$ is:

- *true* if a shortest path between a and b exists and has weight w ;
- *false* otherwise.

Proof

We will compute the *ult*-well-founded model of P by an alternating fixpoint computation using the sequences $(I_n)_{n \in \mathbb{N}}$ and $(J_n)_{n \in \mathbb{N}}$ which are defined by mutual induction:

- $I_0 = \perp$;
- $J_n = St_{\Phi_P}^\uparrow(I_n) = \text{lfp}(\Phi_P^2(I_n, \cdot))$;
- $I_n = St_{\Phi_P}^\downarrow(J_{n-1}) = \text{lfp}(\Phi_P^1(\cdot, J_{n-1}))$.

We now construct this sequence until we reach a fixpoint.

1. We show that $J_0 = \text{lfp}(\Phi_P^2(\perp, \cdot)) = C \cup S$ where

$$\begin{aligned} C &= \{cp(a, b, w) \mid \text{there is a path from } a \text{ to } b \text{ with weight } w\} \\ S &= \{sp(a, b, w) \mid \text{there is a path from } a \text{ to } b \text{ with weight } w\}. \end{aligned}$$

In the first iteration $\Phi_P^2(\perp, \perp) = T_{P, \mathcal{D}}^{agg}(\perp) = C_1$ where

$$C_1 = \{cp(a, b, w) \mid \text{edge}(a, b, w) \in \text{edge}^{\mathcal{D}}\}.$$

In the next iteration, the value of $cp/3$ will not change because the value of $sp/3$ has not changed. On the other hand $cp/3$ has changed, so $sp/3$ will change now. First, let us compute the value of the set expression for values a and b :

$$\llbracket \{c \mid cp(a, b, c)\} \rrbracket_{(\perp, C_1)} = (\emptyset, W_{ab})$$

where $W_{a,b} = \{w \mid (a, b, w) \in \text{edge}^{\mathcal{D}}\}$. i.e., there is a path between a and b of length 1. According to Table 2 in Section 7.2, it holds that $\text{ult}(\text{MIN})^2((\emptyset, W_{ab}), x)$ is true for all $x \in W_{ab}$. So, we obtain $\Phi_P^2(\perp, C_1) = C_1 \cup S_1$ where

$$S_1 = \{sp(a, b, w) \mid \text{edge}(a, b, w) \in \text{edge}^{\mathcal{D}}\}.$$

Continuing this process we can show by induction that for every positive integer number $n > 0$ we have

$$\begin{aligned} \Phi_P^2(\perp, \cdot) \uparrow (2n-1) &= \bigcup_{1 \leq i \leq n} C_i \cup \bigcup_{1 \leq i < n} S_i \\ \Phi_P^2(\perp, \cdot) \uparrow (2n) &= \bigcup_{1 \leq i \leq n} C_i \cup \bigcup_{1 \leq i \leq n} S_i \end{aligned}$$

where

$$\begin{aligned} C_i &= \{cp(a, b, w) \mid \text{there is a path from } a \text{ to } b \text{ of length } i \text{ and weight } w\} \\ S_i &= \{sp(a, b, w) \mid \text{there is a path from } a \text{ to } b \text{ of length } i \text{ and weight } w\}. \end{aligned}$$

At the first limit ordinal ω we have

$$\Phi_P^2(\perp, \cdot) \uparrow \omega = \bigcup_{i \in \mathbb{N}} C_i \cup \bigcup_{i \in \mathbb{N}} S_i = C \cup S.$$

Hence, after this first step, we have computed in J_0 all possible path weights between any two pairs of points a and b .

2. Next we show that $I_1 = \text{lfp}(\Phi_P^1(\cdot, C \cup S)) = CE \cup SP$ where:

$$\begin{aligned} SP &= \{sp(a, b, w) \mid \text{there is a shortest path from } a \text{ to } b \text{ of weight } w\} \\ CE &= \{cp(a, b, w) \mid (a, b, w) \in \text{edge}^{\mathcal{D}} \text{ or} \\ &\quad \exists c, w_1: sp(a, c, w_1) \in SP \text{ and } (c, b, w - w_1) \in \text{edge}^{\mathcal{D}}\} \end{aligned}$$

Define also the following sets for every $i > 1$:

$$\begin{aligned} SP_i &= \{sp(a, b, w) \mid \text{there is a shortest path from } a \text{ to } b \text{ of length } i \text{ and weight } w\} \\ CE_i &= \{cp(a, b, w) \mid (a, b, w) \in \text{edge}^{\mathcal{D}} \text{ or} \\ &\quad i > 1 \text{ and } \exists c, w_1: sp(a, c, w_1) \in SP_{i-1} \text{ and } (c, b, w - w_1) \in \text{edge}^{\mathcal{D}}\} \end{aligned}$$

Note that $SP = \bigcup_{i \in \mathbb{N}} SP_i$ and $CE = \bigcup_{i \in \mathbb{N}} CE_i$.

For the first iteration we verify that

$$\Phi_P^1(\perp, C \cup S) = C_1 = CE_1.$$

To see why this is the case, we compute the value of the set expression:

$$\llbracket \{c \mid cp(a, b, c)\} \rrbracket_{(\perp, C \cup S)} = (\emptyset, A_{ab})$$

where

$$A_{ab} = \{w \mid \text{there is a path from } a \text{ to } b \text{ of length } w\}.$$

Further, $\text{ult}(\text{MIN})^1((\emptyset, A_{ab}), w)$ is false for every weight w (see Table 2 of Section 7.2). So, the interpretation of $sp/3$ will be the empty set. The interpretation of $cp/3$ will be the same as the edge relation. Hence we obtain the set CE_1 .

On the next iteration only the value of $sp/3$ will change. We have

$$\llbracket \{c \mid cp(a, b, c)\} \rrbracket_{(CE_1, C \cup S)} = (W_{ab}, A_{ab})$$

and $((W_{ab}, A_{ab}), w) \in \text{ult}(\text{Min})^1$ if $w \in W_{ab}$ and the shortest path between a and b has weight w . So

$$\Phi_P^1(CE_1, C \cup S) = CE_1 \cup SP_1.$$

On the next iteration, the $cp/3$ relation becomes the composition of SP_1 with the $\text{edge}^{\mathcal{D}}$ relation:

$$\Phi_P^1(CE_1 \cup SP_1, C \cup S) = CE_1 \cup CE_2 \cup SP_1.$$

In the next step we compute $CE_1 \cup CE_2 \cup SP_1 \cup SP_2$, i.e. we obtain all shortest paths of at most length 2. Continuing this process we obtain a fixpoint which is

$$\Phi_P^1(CE \cup SP, C \cup S) = CE \cup SP.$$

At this stage, we have found all shortest paths.

3. Next, we show that $J_1 = \text{lfp}(\Phi_P^2(CE \cup SP, \cdot)) = CE \cup SP$, i.e. $\Phi_P^2(CE \cup SP, CE \cup SP) = CE \cup SP$.

It is straightforward to verify that $cp(a, b, w) \in \Phi_P^2(CE \cup SP, CE \cup SP)$ if and only if $cp(a, b, w) \in CE$. As for $sp(a, b, w)$, we first have to consider the

value of the set expression

$$\llbracket \{c \mid cp(a, b, c)\} \rrbracket_{(CE \cup SP, CE \cup SP)} = (B_{ab}, B_{ab})$$

where

$$B_{ab} = W_{ab} \cup \{w \mid \exists c, w_1 : sp(a, c, w_1) \in SP \wedge (c, b, w - w_1) \in edge^{\mathcal{D}}\}.$$

Either there is no path from a to b , in which case B_{ab} is empty, and its minimum undefined. In this case, no $sp(a, b, w)$ atom is derived. Or, there is a path from a to b . Then by the assumption of the proposition, there is a shortest path from a to b , say with weight w . This minimal path is either an edge from a to b or it is an extension of a shortest path from a to some vertex c . In both cases, w belongs to the set B_{ab} and is its least element. In this case, $sp(a, b, w)$ is derived. Hence

$$\Phi_P^2(CE \cup SP, CE \cup SP) = CE \cup SP.$$

4. Since $J_1 = I_1$, it follows that $I_2 = J_1 = I_1$ and that we have reached a fixpoint which is the two-valued *ult*-well-founded model that was to be proven.

□

The following example shows that the condition in the proposition is essential for the proof.

Example Appendix A.1

Consider the following graph $\{(A, A, -1), (A, B, 0)\}$. It is easy to see that there are no shortest paths because of the cycle in A . Hence, $SP = \emptyset$.

We compute the *ult*-well-founded model of P by an alternating fixpoint computation. The first three steps are exactly as in the proof. Things only change in the computation of J_1 . For this step, the proof exploited the fact that all connected pairs have a shortest path, but this assumption does not hold anymore. We have:

- $I_0 = \perp$;
- $J_0 = \{sp(A, A, -n-1), sp(A, B, -n), cp(A, A, -n-1), cp(A, B, -n) \mid n \in \mathbb{N}\}$.
This describes all paths in the graph. Since there are no shortest paths, in the next step, the computed SP is empty and $CE = \{cp(A, A, -1), cp(A, B, 0)\}$ is just a copy of the edge relation.
- $I_1 = \{cp(A, A, -1), cp(A, B, 0)\} = CE$.
- $J_1 = \text{lfp}(\Phi_P^2(CE, \cdot))$. The computation is entirely similar to the fixpoint computation of J_0 (see the proof of the proposition). Define

$$C_i = \{cp(A, A, -n-1), cp(A, B, -n) \mid n \in [0, i]\}$$

$$S_i = \{sp(A, A, -n), sp(A, B, -n+1) \mid n \in [0, i]\}.$$

Note that $CE = C_0 \cup S_0$. First, we compute $\Phi_P^2(CE, CE)$. Since there are no true sp atoms, the computed cp atoms correspond to the edges. Hence, cp remains unchanged. As for sp , its rule derives the atoms $sp(A, A, -1)$ and $sp(A, B, 0)$, i.e. we obtain $C_0 \cup S_1$. In the next iteration, since cp did not

change, sp remains unaltered. Now cp is extended by composing S_1 with the edge relation. We obtain C_1 . In the next iteration, cp remain identical, and now sp will be extended to obtain S_2 . In general we have the same fixpoint computation as for J_0 except that we start at CE rather than at \perp . It holds that

$$\begin{aligned}\Phi_P^2(CE, \cdot) \uparrow (2n) &= C_n \cup S_n \\ \Phi_P^2(CE, \cdot) \uparrow (2n+1) &= C_n \cup S_{n+1}\end{aligned}$$

The limit of this sequence, J_1 , is equal to J_0 . Therefore, I_2 will be equal to I_1 , so we reached the well-founded fixpoint which is (I_1, J_0) . Since $I_1 \neq J_0$, this is a three-valued model. For example, for each negative integer n , the atom $sp(A, B, n)$ is unknown. \square