TLP **7** (3): 301–353, 2007. © 2007 Cambridge University Press
doi:10.1017/S1471068406002973 Printed in the United Kingdom

Well-founded and stable semantics of logic programs with aggregates

NIKOLAY PELOV, MARC DENECKER and MAURICE BRUYNOOGHE

Department of Computer Science, Katholieke Universiteit Leuven, Belgium
(*e-mail:* {pelov,marcd,maurice}@cs.keleuven.be)

submitted 1 November 2005; revised 14 April 2006; accepted 4 May 2006

Abstract

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.

KEYWORDS: logic programming, aggregates

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

instance:

- the query for the *average* result of students for some exam;
- the property that the 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 general 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 logic programming: Kripke-Kleene semantics (i.e., three-valued completion 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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 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 a formula B' which is equivalent with B in classical logic, is equivalence preserving[1] 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 for programs containing a recursively defined predicate p whose definition also uses reasoning by cases of the form $(p \wedge \ldots) \vee (\neg p \wedge \ldots)$. 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

[1] In the sense of having the same set of models.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 $\langle L, \leqslant \rangle$ is a *poset* if $\leqslant$ is a reflexive, asymmetric and transitive binary relation on L. A poset $\langle L, \leqslant \rangle$ is a *chain* if $\leqslant$ is a total order, i.e., for each $x, y \in L$, either $x \leqslant y$ or $y \leqslant x$. Sometimes, when the order relation $\leqslant$ is clear from the context we denote a poset simply with its domain L.

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

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

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

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

Proposition 2.1
If $O : L \to L$ is a monotone operator and L is a chain-complete poset then there exists an ordinal α such that $O \uparrow^{\alpha} (\bot) = \mathrm{lfp}(O)$. The least such ordinal is called the *closure ordinal* of O and is denoted by ∞.

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 $\langle L, \leqslant \rangle$, its *bilattice* is the structure $\langle L^2, \leqslant, \leqslant_p \rangle$ where for all $x, y, x', y' \in L$,

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

The order $\leqslant$ on L^2 is called the *product order*, while $\leqslant_p$ 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 $x \leqslant y$ and *exact* if $x = y$. The set of consistent pairs is denoted by L^c.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 $[x, y] = \{z \in L \mid x \leqslant z \leqslant y\}$. More precise pairs approximate fewer elements: $(x, y) \leqslant_p (x_1, y_1)$ 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 $\leqslant_p$ is chain-complete on L^c. Hence, any $\leqslant$-monotone or $\leqslant_p$-monotone operator on L^c has a least fixpoint in the $\leqslant$ or $\leqslant_p$ order. Notice also that $(\bot, \top)$ is the $\leqslant_p$-least element of L^c while the $\leqslant_p$-maximal elements of L^c are precisely the set of exact elements.

Example 2.1
Consider the lattice $\mathcal{TWO} = \{\mathbf{f}, \mathbf{t}\}$ of classical truth values ordered as $\mathbf{f} < \mathbf{t}$. We denote the set $\mathcal{TWO}^c$ of consistent approximations of $\mathcal{TWO}$ with $\mathcal{THREE}$.

The set $\mathcal{THREE}$ corresponds to the set of truth values $\{\tilde{\mathbf{f}}, \tilde{\mathbf{u}}, \tilde{\mathbf{t}}\}$ 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 $\tilde{\mathbf{f}}$ and $\tilde{\mathbf{t}}$, while $(\mathbf{f}, \mathbf{t})$ corresponds to $\tilde{\mathbf{u}}$, called *undefined*. The product order on $\mathcal{THREE}$ corresponds to the truth order $\tilde{\mathbf{f}} < \tilde{\mathbf{u}} < \tilde{\mathbf{t}}$. The precision order on $\mathcal{THREE}$ corresponds to the order $\tilde{\mathbf{u}} <_p \tilde{\mathbf{f}}, \tilde{\mathbf{u}} <_p \tilde{\mathbf{t}}$, and is sometimes called the knowledge order.

We can define logical connectives in $\mathcal{THREE}$ 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 $\leqslant$. The negation operator is defined as $\neg(x, y) = (\neg y, \neg x)$. In particular, $\neg\tilde{\mathbf{f}} = \tilde{\mathbf{t}}$, $\neg\tilde{\mathbf{t}} = \tilde{\mathbf{f}}$, and $\neg\tilde{\mathbf{u}} = \tilde{\mathbf{u}}$. 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 $O : L \to L$ be an operator on a complete lattice $\langle L, \leqslant \rangle$. We say that $A : L^c \to L^c$ 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 $\leqslant_p$-monotone.

We denote the projections of an approximating operator $A : L^c \to L^c$ 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 $[x, y] = \{z \mid x \leqslant z \leqslant y\}$. It is easy to see that for any $z \in [x, y]$, $A(x, y) \leqslant_p A(z, z) = (O(z), O(z))$ which means that the set $O([x, y]) = \{O(z) \mid z \in [x, y]\}$ 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 $\leqslant_p$-monotonicity of A guarantees that A has a least fixpoint called the *Kripke-Kleene* fixpoint of A and denoted by $KK(A)$. This fixpoint approximates all fixpoints of O, i.e., if $x = O(x)$ then $KK(A) \leqslant_p (x, x)$.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 $x \in [\bot, b]$ to $A^1(x, b)$. The operator $A^1(\cdot, b)$ is monotone but as a function from $[\bot, b]$ to L, in general it is not internal in $[\bot, b]$. Similarly, for a fixed element $a \in L$, the operator $A^2(a, \cdot) : [a, \top] \to L$ is monotone, but in general is not internal in $[a, \top]$.

Definition 2.2
Let $\langle L, \leqslant \rangle$ be a complete lattice. The *upper stable operator* $St_A^{\uparrow} : L \to L$ is defined as

$$St_A^{\uparrow}(a) = glb(\{x \in [a, \top] : A^2(a, x) \leqslant x\})$$

and the *lower stable operator* $St_A^{\downarrow} : L \to L$ is defined as:

$$St_A^{\downarrow}(b) = glb(\{x \in [\bot, b] : A^1(x, b) \leqslant x\}).$$

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 $A^2(a, \cdot)$ is internal in $[a, \top]$ then due to its monotonicity, $St_A^{\uparrow}(a)$ is its least fixpoint. The lower 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 $[\bot, b]$, then $St_A^{\downarrow}(b)$ is its least fixpoint.

Definition 2.3
The *stable revision operator* $\mathcal{S}t_A : L^2 \to L^2$ where $\langle L, \leqslant \rangle$ is a complete lattice is defined as follows:

$$\mathcal{S}t_A(a, b) = (St_A^{\downarrow}(b), St_A^{\uparrow}(a)).$$

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 $(a, b) \leqslant_p A(a, b)$.
- A pair $(a, b) \in L^c$ is *A-prudent*, if $a \leqslant St_A^{\downarrow}(b)$.

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 $x_0 = \bot, x_1 = A^1(x_0, b), \ldots, x_{i+1} = A^1(x_i, b), \ldots$ until a fixpoint is reached. In each stage, we use A to approximate $O([x_i, b])$ from below, i.e., by setting $x_{i+1} := A^1(x_i, b) \leqslant O(z)$, 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 $y_0 = a, y_1 = A^2(a, y_0), \ldots, y_{i+1} = A^2(a, y_i), \ldots$. The goal is to eliminate non-minimal, *non-reachable* fixpoints of O above the current lower estimate a. Assuming that $a \leqslant O(a) (= A^2(a, a) = y_1)$, all points in $[a, O(a)]$ are considered reachable. On the

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 $St^{\uparrow}_A(a)$ 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 $(\bot, \top)$, is chain-complete, and the stable revision operator is an internal, $\leqslant_p$-monotone operator in L^{rp}. It follows that this operator has a least fixpoint, called the *well-founded* fixpoint of A and denoted by $WF(A)$. All consistent fixpoints of $\mathcal{S}t_A$ are A-reliable and A-prudent. They are called *stable fixpoints* of A and they are $\leqslant$-minimal fixpoints of A. The subset of exact stable fixpoints is denoted by $ST(A)$. 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 $\mathrm{lfp}(A^1(\cdot, x)) = x$. We have the following lemma.

Lemma 2.1
A stable fixpoint of A is a minimal pre-fixpoint of O.

The inverse, however, is not true: not every minimal pre-fixpoint of O is a stable fixpoint of A.

Proof
Let x be a stable fixpoint of A and y a pre-fixpoint of O such that $y \leqslant x$. By anti-monotonicity of A^1 in the second argument, it holds that $A^1(y, x) \leqslant A^1(y, y) = O(y) \leqslant y$. Hence, y is a pre-fixpoint of $A^1(\cdot, x)$ and since x is the least fixpoint of this monotone operator, this implies that $x \leqslant y$. □

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

Theorem 2.1
If $A \leqslant_p B$, then $KK(A) \leqslant_p KK(B)$, $WF(A) \leqslant_p WF(B)$ 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:

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

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 2.2 (*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.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Theorem 2.3 (*Denecker et al. 2004*)
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 power-set 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 $\mathrm{F} : \wp(D_1) \to D_2$. An *aggregate relation* is any relation $\mathrm{R} \subseteq \wp(D_1) \times D_2$.

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* $\mathrm{G_F}$ which is the aggregate relation defined as $\mathrm{G_F} = \{(S, d) \mid \mathrm{F}(S) = d\}$.

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 $\langle D, \leqslant \rangle$:

Definition 3.2

- $\textsc{glb} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid S \in \wp(D) \text{ and } d = \textsc{glb}(S)\}$.
- $\textsc{lub} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid S \in \wp(D) \text{ and } d = \textsc{lub}(S)\}$.
- $\textsc{lb} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid \forall x \in S, d \leqslant x\}$.
- $\textsc{ub} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid \forall x \in S, x \leqslant d\}$.
- $\textsc{min} \subseteq \wp(D) \times D$ - defined as $\{(S, d) \mid d \text{ is a minimal element of } S\}$.
- $\textsc{max} \subseteq \wp(D) \times D$ - 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 $\langle D, \leqslant \rangle$ is a complete lattice then GLB and LUB are (graphs of) total aggregate functions. If $\langle D, \leqslant \rangle$ 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 MIN = GLB and MAX = 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 $D_1 \times \cdots \times D_n$ 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 $\wp_F(D)$.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Definition 3.3

- CARD : $\wp_F(D) \to \mathbb{N}$ defined as $\text{CARD}(S) = |S|$, the cardinality of S;
- SUM : $\wp_F(D') \to \mathbb{R}$ defined as $\text{SUM}(S) = \sum_{(x_1,\ldots,x_n)\in S} x_1$;
- PROD : $\wp_F(D') \to \mathbb{R}$ defined as $\text{PROD}(S) = \prod_{(x_1,\ldots,x_n)\in S} x_1$;
- 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 $S \subseteq \mathbb{R} \times D_2 \times \cdots \times D_n$ 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 extended easily to the more general case. An example of an aggregate relation with two set arguments is the generalized quantifier MOST $\subseteq \wp(D) \times \wp(D)$ where $\text{MOST}(A, B)$ expresses that *most A's are B's*. The relation MOST is defined as the set of all pairs of sets (A, B) such that $\text{CARD}(A \cap B) > \text{CARD}(A \setminus B)$.

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 $\text{R} \subseteq \wp(D_1) \times D_2$ with a binary relation $P \subseteq D_2 \times D_3$ is the aggregate relation $\text{R}_P \subseteq \wp(D_1) \times D_3$ defined as:

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

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

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

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

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Table 1. *Monotone aggregate functions on finite sets*

aggregate	D_1	$\langle D', \leqslant \rangle$
CARD	*arbitrary*	$\langle \mathbb{N}, \leqslant \rangle$
SUM	$\mathbb{R}^+$	$\langle \mathbb{R}^+, \leqslant \rangle$
SUM	$\mathbb{R}^-$	$\langle \mathbb{R}^-, \geqslant \rangle$
PROD	$\mathbb{R}^{[1,\infty)}$	$\langle \mathbb{R}^{[1,\infty)}, \leqslant \rangle$
PROD	$\mathbb{R}^{[0,1)}$	$\langle \mathbb{R}^{[0,1)}, \geqslant \rangle$

Definition 3.5
The *subset aggregate* of an aggregate relation $\mathrm{R} \subseteq \wp(D_1) \times D_2$ is the aggregate $\mathrm{R}_{\subseteq} \subseteq \wp(D_1) \times D_2$ defined as:

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

For an aggregate function F, $\mathrm{F}_{\subseteq}$ denotes the subset aggregate of the graph of F. As an example, $\mathrm{CARD}_{\subseteq}(S, d)$ holds if for some subset $S' \subseteq S$, $\mathrm{CARD}(S') = d$. In this case, the two derived aggregates $\mathrm{CARD}_{\subseteq}$ and $\mathrm{CARD}_{\geqslant}$ are identical. This is because there exists a subset $S' \subseteq S$ such that $\mathrm{CARD}(S') = d$ if and only if $\mathrm{CARD}(S) \geqslant d$.

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 $\langle D_2, \leqslant \rangle$ be a poset and $\mathrm{F} : \wp(D_1) \to D_2$ an aggregate function. We say that F is:

- *monotone* if $S_1 \subseteq S_2$ implies $\mathrm{F}(S_1) \leqslant \mathrm{F}(S_2)$;
- *anti-monotone* if $S_1 \subseteq S_2$ implies $\mathrm{F}(S_1) \geqslant \mathrm{F}(S_2)$.

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

Proposition 3.1
Let $\langle D, \leqslant \rangle$ be a complete lattice. The aggregate function $\mathrm{GLB} : \wp(D) \to D$ is anti-monotone with respect to $\leqslant$ and the aggregate function $\mathrm{LUB} : \wp(D) \to D$ is monotone with respect to $\leqslant$.

Proof
Standard result from lattice theory (Davey and Pristley, 1990, Lemma 2.22). □

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 $D_1 \times \cdots \times D_n$ where $n \geqslant 1$. The aggregate functions in Table 1 from $\wp_F(D)$, the set of finite subsets of D, to the poset $\langle D', \leqslant \rangle$ are monotone when D_1 is as given in the table.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Proof
Follows immediately from well-known properties of real numbers. □

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

Definition 3.7
Let $\mathrm{R} \subseteq \wp(D_1) \times D_2$ be an aggregate relation. We say that R is:

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

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

Proposition 3.3
The aggregate relations $\mathrm{LB}, \mathrm{UB} \subseteq \wp(D) \times D$ on a poset $\langle D, \leqslant \rangle$ are anti-monotone.

All other relations defined in Definition 3.2 are neither a monotone nor an 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 $\mathrm{F} : \wp(D_1) \to D_2$ be an aggregate function which is monotone with respect to a partial order relation $\leqslant$ on D_2. Then $\mathrm{F}_{\geqslant}$ and $\mathrm{F}_{>}$ are monotone aggregate relations and $\mathrm{F}_{\leqslant}$ and $\mathrm{F}_{<}$ are anti-monotone aggregate relations.

Proof
$\mathrm{F}_{\geqslant}$: Consider two sets S_1 and S_2 such that $S_1 \subseteq S_2 \subseteq D_1$ and an element $d \in D_2$. Suppose that $\mathrm{F}_{\geqslant}(S_1, d)$ holds. By definition of $\mathrm{F}_{\geqslant}$ this is equivalent to $\mathrm{F}(S_1) \geqslant d$. Since F is monotone we also have that $\mathrm{F}(S_1) \leqslant \mathrm{F}(S_2)$. So, we can conclude $\mathrm{F}(S_2) \geqslant d$ which is equivalent to $\mathrm{F}_{\geqslant}(S_2, d)$.

The monotonicity of $\mathrm{F}_{>}$ and the anti-monotonicity of $\mathrm{F}_{\leqslant}$ and $\mathrm{F}_{<}$ can be proven in a similar fashion. □

As an application of this result we have that the aggregate relations $\mathrm{CARD}_{\geqslant}$ and $\mathrm{CARD}_{>}$ are monotone since the aggregate function CARD is monotone with respect to $\leqslant$.

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

Proposition 3.5
Let $\mathrm{R} \subseteq \wp(D_1) \times D_2$ be an arbitrary aggregate relation.

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 $s_1, \ldots, s_n$ 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* $f : s_1 \times \cdots \times s_n \to w$ where $n \geqslant 0$;
- P is a set of sorted *predicate symbols* $p : s_1 \times \cdots \times s_n$ where $n \geqslant 0$;
- A is a set of sorted *aggregate symbols* $\mathrm{R} : \{s_1 \times \cdots \times s_n\} \times w$ where $n \geqslant 1$.

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 $f : \to w$ a *constant*. An aggregate symbol $\mathrm{R} : \{s_1 \times \cdots \times s_n\} \times w$ denotes an aggregate relation between sets of type $s_1 \times \cdots \times s_n$ 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 $f : s_1 \times \cdots \times s_n \to w \in Func(\Sigma)$ and $t_1, \ldots, t_n$ are terms of type $s_1, \ldots, s_n$ respectively, then $f(t_1, \ldots, t_n)$ is a term of type w.

An *atom* has the form $p(t_1, \ldots, t_n)$ where $p : s_1 \times \cdots \times s_n \in Pred(\Sigma)$ is a predicate symbol and $t_1, \ldots, t_n$ are terms of types $s_1, \ldots, s_n$ 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 $\{s_1 \times \cdots \times s_n\}$ has the form $\{(x_1, \ldots, x_n) \mid \varphi\}$ where φ is an aggregate formula called the *condition* of the expression and for each $i = 1, \ldots, n$, x_i is a variable of sort s_i.

An *aggregate atom* has the form $\mathrm{R}(s, t)$ where $\mathrm{R} : \{s_1 \times \cdots \times s_n\} \times w \in Aggr(\Sigma)$ is an aggregate symbol, s is a set expression of type $\{s_1 \times \cdots \times s_n\}$, 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. We also use $\varphi \subset \psi$ as an abbreviation for $\varphi \vee \neg\psi$.

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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:

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

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:

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

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.

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

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 $\{(x_1, \ldots, x, \ldots, x_n) \mid \varphi\}$ in ψ. An occurrence of x in ψ is *free* if it is not bounded. The set of free variables of ψ, denoted by $Free(\psi)$, 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* $\mathscr{D}$ consists of the following:

- for each sort $s \in S$ a domain $s^{\mathscr{D}}$;
- for each function symbol $f : s_1 \times \cdots \times s_n \to w \in Func(\Sigma)$ a function
$$f^{\mathscr{D}} : s_1^{\mathscr{D}} \times \cdots \times s_n^{\mathscr{D}} \to w^{\mathscr{D}};$$
- for each predicate symbol $p : s_1 \times \cdots \times s_n \in Pred(\Sigma)$ a relation
$$p^{\mathscr{D}} \subseteq s_1^{\mathscr{D}} \times \cdots \times s_n^{\mathscr{D}}.$$
- for each aggregate symbol $\text{R} : \{s_1 \times \cdots \times s_n\} \times w \in Aggr(\Sigma)$ an aggregate relation $\text{R}^{\mathscr{D}} \subseteq \wp(s_1^{\mathscr{D}} \times \cdots \times s_n^{\mathscr{D}}) \times w^{\mathscr{D}}$.

Consider a Σ-formula $\varphi(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$ of sorts $s_1, \ldots, s_n$, respectively and let $d_1, \ldots, d_n$ be elements of $s_1^{\mathscr{D}}, \ldots, s_n^{\mathscr{D}}$, respectively. Then, $\varphi(d_1, \ldots, d_n)$

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 $\Sigma(\mathscr{D})$ and the corresponding set of formulas with $\mathscr{L}^{aggr}_{\Sigma(\mathscr{D})}$.

Definition 4.5
The value $[\![t]\!]_{\mathscr{D}}$ of a ground term t for a Σ-structure $\mathscr{D}$ is defined inductively as follows:

- if t is a domain element d, then $[\![t]\!]_{\mathscr{D}} = d$;
- if t is a constant c, then $[\![t]\!]_{\mathscr{D}} = c^{\mathscr{D}}$;
- if t is a term $f(t_1, \ldots, t_n)$, then $[\![t]\!]_{\mathscr{D}} = f^{\mathscr{D}}([\![t_1]\!]_{\mathscr{D}}, \ldots, [\![t_n]\!]_{\mathscr{D}})$.

In the following definition and in the rest of the paper we often treat a relation $R \subseteq D$ as a function $R : D \to \{\mathbf{f}, \mathbf{t}\}$ 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 $\mathscr{D}$. We define the value $[\![\{\overline{x} \mid \varphi\}]\!]_{\mathscr{D}}$ of a set expression $\{\overline{x} \mid \varphi\}$ and the truth value $\mathscr{H}_{\mathscr{D}}(\psi)$ of an aggregate formula ψ by simultaneous induction.

The value $[\![\{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}]\!]_{\mathscr{D}}$ of a set expression is the set

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

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

$$\begin{array}{lll}
\mathscr{H}_{\mathscr{D}}(p(t_1, \ldots, t_n)) & = p^{\mathscr{D}}([\![t_1]\!]_{\mathscr{D}}, \ldots, [\![t_n]\!]_{\mathscr{D}}) & \\
\mathscr{H}_{\mathscr{D}}(\mathrm{R}(\{\overline{x} \mid \varphi\}, t)) & = \mathrm{R}^{\mathscr{D}}([\![\{\overline{x} \mid \varphi\}]\!]_{\mathscr{D}}, [\![t]\!]_{\mathscr{D}}) & \\
\mathscr{H}_{\mathscr{D}}(\neg\varphi) & = \neg\mathscr{H}_{\mathscr{D}}(\varphi) & \\
\mathscr{H}_{\mathscr{D}}(\varphi \vee \psi) & = \mathscr{H}_{\mathscr{D}}(\varphi) \vee \mathscr{H}_{\mathscr{D}}(\psi) & \\
\mathscr{H}_{\mathscr{D}}(\varphi \wedge \psi) & = \mathscr{H}_{\mathscr{D}}(\varphi) \wedge \mathscr{H}_{\mathscr{D}}(\psi) & \\
\mathscr{H}_{\mathscr{D}}(\exists x \varphi(x)) & = \bigvee_{d \in s^{\mathscr{D}}} \mathscr{H}_{\mathscr{D}}(\varphi(d)) & \text{(where } s \text{ the sort of } x\text{)} \\
\mathscr{H}_{\mathscr{D}}(\forall x \psi(x)) & = \bigwedge_{d \in s^{\mathscr{D}}} \mathscr{H}_{\mathscr{D}}(\psi(d)) & \text{(where } s \text{ the sort of } x\text{)}
\end{array}$$

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

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:

$$\Sigma = \langle \{pos, nat\};\ \{+ : nat \times nat \to nat, * : nat \times nat \to nat, / : nat \times nat \to nat,$$
$$Dim : nat, Mc : nat, f : pos \times pos \to nat\};\ \emptyset;\ \{\mathrm{SUM} : \{nat\} \times nat\} \rangle$$

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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:

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

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

5 Aggregate Programs

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

Given are 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 $\mathscr{D}$ 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 $+$, $*$, $-$,... on these sorts interpreted as the corresponding operations on numbers;
- the standard predicate symbols $=$, $\leqslant$,... on these sorts interpreted as the corresponding relations on numbers;
- all aggregate symbols defined in Section 3.1: CARD, MIN, MAX, SUM, ...

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 $s^{\mathscr{D}}$ of all sorts $s \in S_d$ and the interpretation $f^{\mathscr{D}}$ of all function symbols $f : s_1 \times \cdots \times s_n \rightarrow s \in F_d$ are defined by simultaneous induction as follows:

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 $\mathscr{D}$.

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. Note that φ may contain universal quantifiers. 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 $\mathscr{D}$-*base* $base_{\mathscr{D}}(\Pi)$ of Π is defined as

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

The semantics of an aggregate program will be defined in the collection of $\Sigma(\Pi)$-structures extending $\mathscr{D}$. For each subset I of $base_{\mathscr{D}}(\Pi)$, we define the $\Sigma(\Pi)$-structure $\mathscr{D}(I)$ extending $\mathscr{D}$ such that for every atom $A \in base_{\mathscr{D}}(\Pi)$: $\mathscr{H}_{\mathscr{D}(I)}(A) = \mathbf{t}$ if and only if $A \in I$. Clearly, this is a one-to-one correspondence between the subsets of $base_{\mathscr{D}}(\Pi)$ and $\Sigma(\Pi)$-extensions of $\mathscr{D}$. In the rest of the paper, we exploit this correspondence and use subsets of $base_{\mathscr{D}}(\Pi)$, called *interpretations*, to represent $\Sigma(\Pi)$-extensions of $\mathscr{D}$. Sometimes, we also view an interpretation I as a mapping $I : base_{\mathscr{D}}(\Pi) \to \mathscr{TWO}$.

An interpretation I is a *model* of an aggregate program P if I is a model of the first-order theory obtained from P by turning every rule $A \leftarrow \varphi$ into an implication $\forall \bar{x}(A \subset \varphi)$ where $\bar{x}$ are the free variables of A and φ.

The set $\mathscr{I} = \wp(base_{\mathscr{D}}(\Pi))$ forms a complete lattice under the subset order $\subseteq$. This order extends to $\Sigma(\Pi)$-structures as follows: $D(I) \leqslant D(J)$ if and only if $I \subseteq J$.

We introduce the following notation. For any closed defined atom $A = p(t_1, \ldots, t_n)$, $[\![A]\!]_{\mathscr{D}}$ denotes the atom $p([\![t_1]\!]_{\mathscr{D}}, \ldots, [\![t_n]\!]_{\mathscr{D}}) \in base_{\mathscr{D}}(\Pi)$.

Definition 5.1

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

- there exists a rule $A' \leftarrow \varphi' \in P$ with free variables $x_1, \ldots, x_m$ of sorts $s_1, \ldots, s_m$, and
- there exist domain elements $d_1 \in s_1^{\mathscr{D}}, \ldots, d_m \in s_m^{\mathscr{D}}$, and
- $A = [\![A'(d_1, \ldots, d_m)]\!]_{\mathscr{D}}$ and
- $\varphi = \varphi'(d_1, \ldots, d_m)$.

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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

Definition 5.2
The *two-valued immediate consequence operator* $T^{aggr}_{P,\mathscr{D}} : \mathscr{I} \to \mathscr{I}$ of an aggregate program P is defined as:

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

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

As for standard logic programs we have a correspondence between models of an aggregate program P and pre-fixpoints of $T^{aggr}_{P,\mathscr{D}}$.

Proposition 5.1
An interpretation I is a model of an aggregate program P if and only if I is a pre-fixpoint of $T^{aggr}_{P,\mathscr{D}}$, i.e., $T^{aggr}_{P,\mathscr{D}}(I) \leqslant I$.

Proof
The proof is straightforward extension of the proof for standard logic programs. □

Definition 5.3
An interpretation I is a *supported model* of an aggregate program P extending $\mathscr{D}$ if I is a fixpoint of $T^{aggr}_{P,\mathscr{D}}$.

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*)
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. Consider the following aggregate signature:

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

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:

$$accept(x) \leftarrow thr(x, t) \wedge \text{CARD}_{\geqslant}(\{y \mid friend(x, y) \wedge accept(y)\}, t).$$

Consider an instance of the problem with two friends, say A and B. Each one of them accepts the invitation if and only if the other one accepts as well. This is represented by the Σ-structure $\mathscr{D}$ in which $person^{\mathscr{D}} = \{A, B\}$ and in which $friend^{\mathscr{D}}$

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

and $thr^{\mathscr{D}}$ are given by the following table:

$$friend(A, B). \qquad thr(A, 1).$$
$$friend(B, A). \qquad thr(B, 1).$$

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.

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 (2004). 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 $\mathscr{D}$.

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 $\mathrm{R}(\{\overline{x} \mid \theta\}, t)$ in φ such that $\mathrm{R}^{\mathscr{D}}$ 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 an anti-monotone aggregate relation above P (resp. ψ) is even and *negative* if the number of negations and aggregate atoms interpreted with an anti-monotone aggregate relation above P (resp. ψ) is odd.

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 $\mathrm{R}(\{\overline{x} \mid \varphi\}, t)$ there are three cases in which it can be positive. The first one is when $\mathrm{R}^{\mathscr{D}}$ is a monotone aggregate relation and φ is a positive formula. The second case is when $\mathrm{R}^{\mathscr{D}}$ is an anti-monotone aggregate relation and φ is a negative formula. The third one is when $\mathrm{R}^{\mathscr{D}}$ is arbitrary and φ does not contain defined predicates. Similarly, the aggregate atom $\mathrm{R}(\{\overline{x} \mid \varphi\}, t)$ is negative if $\mathrm{R}^{\mathscr{D}}$ is a monotone aggregate relation and

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

φ is negative, $\mathrm{R}^{\mathscr{D}}$ is an anti-monotone aggregate relation and φ is positive, or $\mathrm{R}^{\mathscr{D}}$ 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 $\mathscr{D}$.

Proposition 5.2
Let $\mathscr{D}$ be a Σ-structure and ψ be a closed $\Sigma(\Pi)$-aggregate formula (possibly containing domain elements). For any pair $I \subseteq J \in base_{\mathscr{D}}(\Pi)$, it holds that:

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

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 $\mathrm{R}(\{\overline{x} \mid \varphi(\overline{x})\}, t)$ without free variables. Let $S_I = [\![\{\overline{x} \mid \varphi(\overline{x})\}]\!]_{\mathscr{D}(I)}$ and $S_J = [\![\{\overline{x} \mid \varphi(\overline{x})\}]\!]_{\mathscr{D}(J)}$.

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

1. $\mathrm{R}^{\mathscr{D}}$ is a monotone aggregate relation and $\varphi(\overline{x})$ is a positive formula. For every well-sorted tuple $\overline{d}$, $\varphi(\overline{d})$ is a positive formula as well. By the induction hypothesis, we have that $\mathscr{D}(I) \models \varphi(\overline{d})$ implies $\mathscr{D}(J) \models \varphi(\overline{d})$. Consequently, $S_I \subseteq S_J$. Finally, because $\mathrm{R}^{\mathscr{D}}$ is a monotone aggregate relation, $(S_I, [\![t]\!]_{\mathscr{D}}) \in \mathrm{R}^{\mathscr{D}}$ implies $(S_J, [\![t]\!]_{\mathscr{D}}) \in \mathrm{R}^{\mathscr{D}}$. Thus, $\mathscr{D}(I) \models \psi$ implies $\mathscr{D}(J) \models \psi$.
2. $\mathrm{R}^{\mathscr{D}}$ is an anti-monotone aggregate relation and $\varphi(\overline{x})$ is a negative formula. By the induction hypothesis, for every appropriate tuple $\overline{d}$ of domain elements assigned to $\overline{x}$, we have $\mathscr{D}(J) \models \varphi(\overline{d})$ implies $\mathscr{D}(I) \models \varphi(\overline{d})$. Consequently, $S_J \subseteq S_I$. Finally, because $\mathrm{R}^{\mathscr{D}}$ is an anti-monotone aggregate relation, $(S_I, [\![t]\!]_{\mathscr{D}}) \in \mathrm{R}^{\mathscr{D}}$ implies $(S_J, [\![t]\!]_{\mathscr{D}}) \in \mathrm{R}^{\mathscr{D}}$. Thus, $\mathscr{D}(I) \models \psi$ implies $\mathscr{D}(J) \models \psi$.
3. If φ contains no defined predicates, then $\mathscr{D}(I) \models \psi$ if and only if $\mathscr{D}(J) \models \psi$ if and only if $\mathscr{D} \models \psi$.

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

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 5.1
If P is a definite aggregate program then $T^{aggr}_{P,\mathscr{D}}$ is monotone.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Proof
Follows immediately from Proposition 5.2. □

Definition 5.7
We define the *least fixpoint model* of a definite $\Sigma(\Pi)$-aggregate program P extending $\mathscr{D}$ as the least fixpoint of its immediate consequence operator $T^{aggr}_{P,\mathscr{D}}$.

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 Control*)
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:

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

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

$$\Pi = \{controls : c \times c\}.$$

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

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

For numbers in the interval [0..1], the SUM aggregate function is monotone with respect to $\geqslant$. 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 $T^{aggr}_{P,\mathscr{D}}$ operator which has a least fixpoint I.

We will now show that the least fixpoint of $T^{aggr}_{P,\mathscr{D}}$ 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 by 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 $n \geqslant 0$, 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%.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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

Proposition 5.3
$controls^{\mathrm{lfp}(T^{aggr}_{P,\mathscr{D}})} = C$.

Proof
Let $I_n = T^{aggr}_{P,\mathscr{D}} \uparrow^n (\emptyset)$ for $n \geqslant 0$. We will prove for each $n \geqslant 0$ that $controls^{I_n} = C^n$. Clearly, it follows from this that $controls^{\mathrm{lfp}(T^{aggr}_{P,\mathscr{D}})} = C$.

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

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

$$S = [\![\{(s, z) \mid (a = z \vee controls(a, z)) \wedge owns_stock(z, b, s)\}]\!]_{I_{n-1}}.$$

It is easy to see that if $controls^{I_{n-1}} = C^{n-1}$ 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 $(a, b) \in C^n$. □

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:

$$\Sigma = \langle \{s\};\ \{compl : s \to s\};\ \{open : s\};\ \{\mathrm{GLB}^{\omega}_{\subseteq}, \mathrm{LUB}^{\omega}_{\subseteq} : \{s\} \times s\} \rangle.$$

The Σ-structure $\mathscr{D}$ 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: $compl^{\mathscr{D}}(S) = \mathbb{R} - S$. The aggregate relations GLB^{ω} and LUB^{ω} are the restrictions of GLB and LUB to countable input sets, i.e., for any set $\mathscr{R}$ of sets of real numbers and set S of real numbers, $(\mathscr{R}, S) \in \mathrm{GLB}^{\omega}$ if and only if $|\mathscr{R}| \leqslant \omega$ and $S = \bigcap \mathscr{R}$. The aggregate relations $\mathrm{GLB}^{\omega}_{\subseteq}$ and $\mathrm{LUB}^{\omega}_{\subseteq}$ are obtained by forming the subset aggregates of GLB^{ω} and LUB^{ω} respectively (see Definition 3.5). Then $\mathrm{GLB}^{\omega}_{\subseteq}(\mathscr{R}, S)$ holds if S is the intersection of some countable subset of $\mathscr{R}$. Likewise $\mathrm{LUB}^{\omega}_{\subseteq}(\mathscr{R}, S)$ holds if S is the union of some countable subset of $\mathscr{R}$.

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

$$borel(S) \leftarrow open(S).$$
$$borel(compl(S)) \leftarrow borel(S).$$
$$borel(S) \leftarrow \mathrm{GLB}^{\omega}_{\subseteq}(\{B \mid borel(B)\}, S) \vee \mathrm{LUB}^{\omega}_{\subseteq}(\{B \mid borel(B)\}, S).$$

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Each of these rules is the formal representation of one of the rules in the inductive definition of Borel sets. Since $\text{GLB}^{\omega}_{\subseteq}$ and $\text{LUB}^{\omega}_{\subseteq}$ are monotone aggregate relations (Proposition 3.5) this is a definite aggregate program and it defines a monotone operator $T^{aggr}_{P,\mathscr{D}}$. 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 $T^{aggr}_{P,\mathscr{D}}$. Thus, $borel(d) \in \text{lfp}(T^{aggr}_{P,\mathscr{D}})$ if and only if d is a Borel set.

5.2 *Stratified aggregate programs*

The important class of stratified aggregate programs was already considered by several authors (Mumick *et al.* 1990; Dell'Armi *et al.* 2003; Faber *et al.* 2004). It is a natural extension of the concept of stratified logic program (Apt *et al.* 1988) where aggregates are treated as negative literals.

Definition 5.8
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 positively in the body B of a rule with head p, then $\|q\| \leqslant \|p\|$ 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 $i \geqslant 0$, $\Sigma_i = \Sigma \cup \bigcup_{1 \leqslant j \leqslant i} \Pi_j$.

Assume $i \geqslant 1$ and fix an arbitrary Σ_{i-1}-structure $\mathscr{D}'$ extending $\mathscr{D}$. 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 $T^{aggr}_{P_i,\mathscr{D}'}$ operator. Note that it does not matter whether the aggregates in P_i are monotone or non-monotone, since they do not contain predicates of Π_i.

Definition 5.9
The *standard model* of an aggregate program P extending $\mathscr{D}$ is the interpretation $I = \bigcup_{1 \leqslant i \leqslant \|P\|} I_i$ where the set $\{I_i \mid 1 \leqslant i \leqslant \|P\|\}$ is defined by the following (finite) induction:

$$\begin{aligned} \mathscr{D}_0 &= \mathscr{D}; \\ I_i &= \text{lfp}\left(T^{aggr}_{P_i,\mathscr{D}_{i-1}}\right); \\ \mathscr{D}_i &= \mathscr{D}\left(\bigcup_{1 \leqslant j \leqslant i} I_j\right). \end{aligned}$$

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

Example 5.4 (*Shortest Path*)
Consider the signature of directed weighted graphs

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

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 pPrograms

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

Definition 6.1
The *ultimate approximating operator* $U^{aggr}_{P,\mathscr{D}} : \mathscr{I}^c \to \mathscr{I}^c$ of $T^{aggr}_{P,\mathscr{D}} : \mathscr{I} \to \mathscr{I}$ is defined as:

$$U^{aggr}_{P,\mathscr{D}}(I_1, I_2) = \left(\bigcap_{I \in [I_1, I_2]} T^{aggr}_{P,\mathscr{D}}(I), \bigcup_{I \in [I_1, I_2]} T^{aggr}_{P,\mathscr{D}}(I) \right).$$

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 $U^{aggr}_{P,\mathscr{D}}$ operator.

Example 6.1
Consider the following program with Herbrand universe $\{0, 1\}$:

$$p(0) \leftarrow \text{CARD}(\{x | p(x)\}, 1).$$

Observe that this program has two supported models: $\emptyset$ and $\{p(0)\}$. Also, this is not a definite aggregate program and its immediate consequence operator is non-monotone as can be seen from:

$$T^{aggr}_{P,\mathscr{D}}(\{p(0)\}) = \{p(0)\},$$
$$T^{aggr}_{P,\mathscr{D}}(\{p(0), p(1)\}) = \emptyset.$$

Let us construct the well-founded fixpoint. We start from the pair $(\bot, \top)$. The new upper and lower bounds are obtained by applying the stable revision operator of

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Definition 2.3 on $(\bot, \top) = (\emptyset, \{p(0), p(1)\})$. The new upper bound is the least fixpoint of $(U^{aggr}_{P,\mathscr{D}})^2(\emptyset, \cdot)$. It is easy to see that:

$$(U^{aggr}_{P,\mathscr{D}})^2(\emptyset, \emptyset) = T^{aggr}_{P,\mathscr{D}}(\emptyset) = \emptyset.$$

It follows that $\emptyset$ is a fixpoint of this operator; it is obviously the least fixpoint. Likewise, the new lower bound is the least fixpoint of $(U^{aggr}_{P,\mathscr{D}})^1(\cdot, \{p(0), p(1)\})$. This is $\emptyset$ as well, since:

$$(U^{aggr}_{P,\mathscr{D}})^1(\emptyset, \{p(0), p(1)\}) = \emptyset;$$

Consequently, the well-founded fixpoint is $(\emptyset, \emptyset)$. This represents the two-valued interpretation $\emptyset$ and this is also the unique ultimate stable model of this program.

The ultimate Kripke-Kleene model can be computed in two computation steps:

$$(U^{aggr}_{P,\mathscr{D}})^1(\emptyset, \{p(0), p(1)\}) = \emptyset; \quad (U^{aggr}_{P,\mathscr{D}})^2(\emptyset, \{p(0), p(1)\}) = \{p(0)\}$$
$$(U^{aggr}_{P,\mathscr{D}})^1(\emptyset, \{p(0)\}) = \emptyset; \qquad (U^{aggr}_{P,\mathscr{D}})^2(\emptyset, \{p(0)\}) = \{p(0)\}.$$

The model is the three-valued interpretation $\{p(0)^{\mathfrak{u}}\} = (\emptyset, \{p(0)\})$.

We obtain the following corollary to Theorem 2.3 and Theorem 5.1.

Corollary 6.1
If $T^{aggr}_{P,\mathscr{D}}$ is monotone, then the ultimate well-founded fixpoint of P is the least fixpoint of $T^{aggr}_{P,\mathscr{D}}$ 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 6.1
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 two-valued $\Sigma(\Pi)$-extensions $\mathscr{D}(I)$ of $\mathscr{D}$, then $T^{aggr}_{P,\mathscr{D}} = T^{aggr}_{P',\mathscr{D}}$ and P and P' have the same ultimate Kripke-Kleene model, the same ultimate well-founded model, and the same set of ultimate stable models.

Another result about the set of ultimate stable models is that they are always minimal models. In fact, we can prove such result for the set of stable models associated with any approximating operator of $T^{aggr}_{P,\mathscr{D}}$.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Proposition 6.2
Let P be an aggregate program and A be an approximating operator of $T^{aggr}_{P,\mathscr{D}}$. Each stable model of A is a minimal model of P.

Proof
By Lemma 2.1, every stable model of A is a minimal pre-fixpoint of $T^{aggr}_{P,\mathscr{D}}$ and by Proposition 5.1, the pre-fixpoints of $T^{aggr}_{P,\mathscr{D}}$ are exactly the models of P. □

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^p_2-complete (Denecker *et al.* 2004). For this reason, we will study weaker semantics based on less precise approximations of $T^{aggr}_{P,\mathscr{D}}$.

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. We first illustrate these points with an example.

Example 7.1
We denote a three-valued set by indexing its *certain* elements with $\tilde{\mathbf{t}}$ and its *possible* elements with $\tilde{\mathbf{u}}$. Let us fix the three-valued set $\tilde{S} = \{1^{\tilde{\mathbf{t}}}, 2^{\tilde{\mathbf{u}}}, 3^{\tilde{\mathbf{t}}}, 5^{\tilde{\mathbf{u}}}\}$.

A three-valued aggregate $\widetilde{\text{CARD}}$ of the cardinality aggregate CARD is a mapping from pairs of three-valued sets and natural numbers to $\mathscr{THREE}$. In case of the set $\tilde{S}$, correct values for the set $\tilde{S}$ are the following:

$$\begin{cases} \widetilde{\text{CARD}}(\tilde{S}, n) = \tilde{\mathbf{u}}, \text{ for all } n \in \{2,3,4\} \\ \widetilde{\text{CARD}}(\tilde{S}, n) = \tilde{\mathbf{f}}, \text{ for all natural numbers } n \notin \{2,3,4\} \end{cases}$$

This specifies that the set approximated by $\tilde{S}$ has between two and four elements.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Similarly, correct values in case of $\text{CARD}_{\geqslant}$ are as follows:

$$\begin{cases} \widetilde{\text{CARD}}_{\geqslant}(\tilde{S}, n) = \tilde{\mathbf{t}}, \text{ for all } n \in \{0, 1, 2\} \\ \widetilde{\text{CARD}}_{\geqslant}(\tilde{S}, n) = \tilde{\mathbf{u}}, \text{ for all } n \in \{3, 4\} \\ \widetilde{\text{CARD}}_{\geqslant}(\tilde{S}, n) = \tilde{\mathbf{f}}, \text{ for all natural numbers } n \notin \{0, \ldots, 4\} \end{cases}$$

This specifies that each set approximated by $\tilde{S}$ certainly has more than zero, one and two elements, and has possibly more than three or four elements, but definitely has not more than five elements or more. A weaker but still correct value for $\tilde{S}$ would be:

$$\begin{cases} \widetilde{\text{CARD}}_{\geqslant}(\tilde{S}, 0) = \tilde{\mathbf{t}}, \\ \widetilde{\text{CARD}}_{\geqslant}(\tilde{S}, n) = \tilde{\mathbf{u}}, \text{ for all } n \in \{1, 2, 3, 4, 5, 6\} \\ \widetilde{\text{CARD}}_{\geqslant}(\tilde{S}, n) = \tilde{\mathbf{f}}, \text{ for all natural numbers } n > 6 \end{cases}$$

which specifies that the set approximated by $\tilde{S}$ has certainly more than 0 elements, possibly more than one to six elements and certainly not more than seven elements or more.

We now formalize the notions of three-valued aggregate relations and structures.

Definition 7.1 (*Three-valued Aggregate Relations*)
A *three-valued aggregate relation* is a function $\mathscr{R} : \wp(D_1)^c \times D_2 \to \mathscr{THREE}$ which satisfies:

- $\leqslant_p$-*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 $\tilde{S}_1 \leqslant_p \tilde{S}_2$ then $\mathscr{R}(\tilde{S}_1, d) \leqslant_p \mathscr{R}(\tilde{S}_2, d)$;
- *exactness*: for every exact (two-valued) set $S \in \wp(D_1)$ and for every $d \in D_2$, $\mathscr{R}((S, S), d) \in \mathscr{TWO}$.

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 $\leqslant_p$-monotonicity and exactness conditions for all set arguments.

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

Recall that $\mathscr{THREE} = \mathscr{TWO}^c$. It follows that a three-valued aggregate relation $\mathscr{R} : \wp(D_1)^c \times D_2 \to \mathscr{THREE}$ is completely determined by the pair $(\mathscr{R}^1, \mathscr{R}^2)$ of its projections on the first and second component. These projections are relations $\mathscr{R}^1, \mathscr{R}^2 \subseteq \wp(D_1)^c \times D_2$ such that $\mathscr{R}^1 \subseteq \mathscr{R}^2$ and $\mathscr{R}^1, \mathscr{R}^2$ coincide on two-valued sets[2]. We will frequently define a three-valued aggregate relation $\mathscr{R}$ by defining $\mathscr{R}^1$ and $\mathscr{R}^2$ separately.

[2] Equivalently, $\mathscr{R}^1, \mathscr{R}^2$ are functions $\wp(D_1)^c \times D_2 \to \mathscr{TWO}$ which coincide on two-valued sets and such that $\mathscr{R}^1 \leqslant \mathscr{R}^2$.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Definition 7.2 (*Three-valued Structure*)
Let Σ be an aggregate signature. A *three-valued* Σ*-structure* $\tilde{\mathscr{D}}$ consists of the following:

- for each sort $s \in S$ a domain $s^{\tilde{\mathscr{D}}}$;
- for each function symbol $f : s_1 \times \cdots \times s_n \to w \in Func(\Sigma)$ a function
$$f^{\tilde{\mathscr{D}}} : s_1^{\tilde{\mathscr{D}}} \times \cdots \times s_n^{\tilde{\mathscr{D}}} \to w^{\tilde{\mathscr{D}}};$$
- for each predicate symbol $p : s_1 \times \cdots \times s_n \in Pred(\Sigma)$ a *three-valued relation*
$$p^{\tilde{\mathscr{D}}} : s_1^{\tilde{\mathscr{D}}} \times \cdots \times s_n^{\tilde{\mathscr{D}}} \to \mathscr{THREE}.$$
- for each aggregate symbol $\mathrm{R} : \{s_1 \times \cdots \times s_n\} \times w \in Aggr(\Sigma)$ a *three-valued aggregate relation*
$$\mathrm{R}^{\tilde{\mathscr{D}}} : \wp((s_1^{\tilde{\mathscr{D}}})^c \times \cdots \times (s_n^{\tilde{\mathscr{D}}})^c) \times w^{\tilde{\mathscr{D}}} \to \mathscr{THREE}.$$

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

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

Definition 7.3
For all three-valued aggregate relations $\mathscr{R}_1, \mathscr{R}_2 : \wp(D_1)^c \times D_2 \to \mathscr{THREE}$, define $\mathscr{R}_1 \leqslant_p \mathscr{R}_2$ if $\mathscr{R}_1(\tilde{S}, d) \leqslant_p \mathscr{R}_2(\tilde{S}, d)$ 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 $\tilde{\mathscr{D}}_1$ and $\tilde{\mathscr{D}}_2$, define $\tilde{\mathscr{D}}_1 \leqslant_p \tilde{\mathscr{D}}_2$ if $\tilde{\mathscr{D}}_1$ and $\tilde{\mathscr{D}}_2$ have the same domain, the same interpretations of sort and function symbols, for each predicate symbol $p \in Pred(\Sigma)$, $p^{\tilde{\mathscr{D}}_1} \leqslant_p p^{\tilde{\mathscr{D}}_2}$ and for each aggregate symbol $\mathrm{R} \in Aggr(\Sigma)$, $\mathrm{R}^{\tilde{\mathscr{D}}_1} \leqslant_p \mathrm{R}^{\tilde{\mathscr{D}}_2}$.

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

Definition 7.4 (*Three-valued valuation and truth functions*)
Let Σ be an aggregate signature and $\tilde{\mathscr{D}}$ be a three-valued Σ-structure. We define the *three-valued valuation function* $[\![\cdot]\!]_{\tilde{\mathscr{D}}}$ for set expressions and the *three-valued truth function* $\mathscr{H}_{\tilde{\mathscr{D}}}$ for aggregate formulas by simultaneous induction.

Let $\{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}$ be a set expression of type $\{s_1 \times \cdots \times s_n\}$. The value $[\![\{(x_1, \ldots, x_n) \mid \varphi\}]\!]_{\tilde{\mathscr{D}}}$ is the three-valued set $\tilde{S}$ defined as:
$$\tilde{S}(d_1, \ldots, d_n) = \mathscr{H}_{\tilde{\mathscr{D}}}(\varphi(d_1, \ldots, d_n))$$
for every $(d_1, \ldots, d_n) \in s_1^{\tilde{\mathscr{D}}} \times \cdots \times s_n^{\tilde{\mathscr{D}}}$.

The *three-valued truth function* for first-order aggregate formulas $\mathscr{H}_{\tilde{\mathscr{D}}}(\cdot) : \mathscr{L}^{aggr}_{\Sigma(\tilde{\mathscr{D}})} \to \mathscr{THREE}$ 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 $\mathscr{H}_{\tilde{\mathscr{D}}}$ is monotone with respect to the precision order $\leqslant_p$ on three-valued interpretations.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Proposition 7.1
Let $\tilde{\mathscr{D}}_1$ and $\tilde{\mathscr{D}}_2$ be three-valued Σ-structures. If $\tilde{\mathscr{D}}_1 \leqslant_p \tilde{\mathscr{D}}_2$ then for every Σ-aggregate formula φ, $\mathscr{H}_{\tilde{\mathscr{D}}_1}(\varphi) \leqslant_p \mathscr{H}_{\tilde{\mathscr{D}}_2}(\varphi)$.

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

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

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. □

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 by $\tilde{\mathscr{D}}(\tilde{I})$ where $\tilde{I} : base_{\tilde{\mathscr{D}}}(\Pi) \to \mathscr{THREE}$ gives the (three-valued) interpretation of the predicates in Π.

We now extend Definition 5.2 of the immediate consequence operator for aggregate programs to the three-valued case.

Definition 7.5
The *three-valued immediate consequence operator* $\Phi^{aggr}_{P,\tilde{\mathscr{D}}} : \mathscr{I}^c \to \mathscr{I}^c$ for an aggregate program P maps any three-valued interpretation $\tilde{I}$ to a three-valued interpretation $\tilde{I}' = \Phi^{aggr}_{P,\tilde{\mathscr{D}}}(\tilde{I})$ such that for each ground defined atom $A \in base_{\tilde{\mathscr{D}}}(\Pi)$:

$$\tilde{I}'(A) = \bigvee \{\mathscr{H}_{\tilde{\mathscr{D}}(\tilde{I})}(\varphi) \mid A \leftarrow \varphi \in inst_{\tilde{\mathscr{D}}}(P)\}.$$

Or, the truth value of a defined atom A in $\tilde{I}'$ is the greatest of all truth values of bodies of rule instances with A in the head.

Proposition 7.3
If $\tilde{\mathscr{D}}$ is a three-valued structure approximating $\mathscr{D}$ then $\Phi^{aggr}_{P,\tilde{\mathscr{D}}}$ is an approximating operator of $T^{aggr}_{P,\mathscr{D}}$.

Proof
Follows from Proposition 7.1 and Proposition 7.2. □

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Definition 7.6
Given a three-valued structure $\tilde{\mathscr{D}}$, the *$\tilde{\mathscr{D}}$-Kripke-Kleene model*, the *$\tilde{\mathscr{D}}$-well-founded model* and the set of *$\tilde{\mathscr{D}}$-stable models* of an aggregate program P are defined as the Kripke-Kleene, well-founded and the set of exact stable fixpoints of the $\Phi^{aggr}_{P,\tilde{\mathscr{D}}}$ operator.

Example 7.2
Reconsider the program of Example 6.1 with Herbrand universe $\{0,1\}$:

$$p(0) \leftarrow \text{CARD}(\{x|p(x)\},1).$$

We will show that, for an appropriate value of the three-valued aggregate, its standard well-founded model is identical to its ultimate well-founded model.

To compute stable and well-founded models, we need to choose a three-valued aggregate $\widetilde{\text{CARD}}$ approximating CARD. Let us assume that

$$\widetilde{\text{CARD}}(\{0^{\tilde{\mathbf{u}}},1^{\tilde{\mathbf{u}}}\},1) = \widetilde{\text{CARD}}((\emptyset,\{0,1\}),1) = \tilde{\mathbf{u}} = (\mathbf{f},\mathbf{t})$$

$$\widetilde{\text{CARD}}(\{0^{\tilde{\mathbf{u}}}\},1) = \widetilde{\text{CARD}}((\emptyset,\{0\}),1) = \tilde{\mathbf{u}} = (\mathbf{f},\mathbf{t}).$$

Then the following assignments can be computed easily:

$$\begin{array}{lll}
(\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^1(\{p(0)^{\tilde{\mathbf{u}}},p(1)^{\tilde{\mathbf{u}}}\}) & = (\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^1((\emptyset,\{p(0),p(1)\})) & = \emptyset;\\
(\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^2(\{p(0)^{\tilde{\mathbf{f}}},p(1)^{\tilde{\mathbf{f}}}\}) & = (\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^2((\emptyset,\emptyset)) & = \emptyset;\\
(\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^2(\{p(0)^{\tilde{\mathbf{u}}},p(1)^{\tilde{\mathbf{u}}}\}) & = (\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^2((\emptyset,\{p(0),p(1)\})) & = \{p(0)\};\\
(\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^1(\{p(0)^{\tilde{\mathbf{u}}}\}) & = (\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^1((\emptyset,\{p(0)\})) & = \emptyset;\\
(\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^2(\{p(0)^{\tilde{\mathbf{u}}}\}) & = (\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^2((\emptyset,\{p(0)\})) & = \{p(0)\}.
\end{array}$$

These assignments are the same as for the ultimate approximation $U^{aggr}_{P,\mathscr{D}}$. It follows that the empty set $\emptyset$ is the $\tilde{\mathscr{D}}$-well-founded model and the unique $\tilde{\mathscr{D}}$-stable model of this program and $\{p(0)^{\tilde{\mathbf{u}}}\}$ is the $\tilde{\mathscr{D}}$-Kripke-Kleene model.

For logic programs without aggregates each $\Phi^{aggr}_{P,\tilde{\mathscr{D}}}$ 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 $\tilde{\mathscr{D}}$ 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 $\tilde{\mathscr{D}}_1$ and $\tilde{\mathscr{D}}_2$ and for every three-valued interpretation $\tilde{I}$, if $\tilde{\mathscr{D}}_1 \leqslant_p \tilde{\mathscr{D}}_2$ then $\Phi^{aggr}_{P,\tilde{\mathscr{D}}_1}(\tilde{I}) \leqslant_p \Phi_{P,\tilde{\mathscr{D}}_2}(\tilde{I})$.

So, by Theorem 2.1 we obtain the following result.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Theorem 7.1
Let P be an aggregate program and $\tilde{\mathscr{D}}_1$ and $\tilde{\mathscr{D}}_2$ be two three-valued Σ-structures such that $\tilde{\mathscr{D}}_1 \leqslant_p \tilde{\mathscr{D}}_2$. Then:

- the $\tilde{\mathscr{D}}_1$-Kripke-Kleene model of P is less precise (in the $\leqslant_p$ order) than the $\tilde{\mathscr{D}}_2$-Kripke-Kleene model of P;
- the $\tilde{\mathscr{D}}_1$-well-founded model of P is less precise (in the $\leqslant_p$ order) than the $\tilde{\mathscr{D}}_2$-well-founded model of P;
- every $\tilde{\mathscr{D}}_1$-stable model is a $\tilde{\mathscr{D}}_2$-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 $\tilde{\mathscr{D}}$-well-founded model of an aggregate program with monotone immediate consequence operator is not necessarily its least fixpoint. For example, 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 7.5
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 $\tilde{\mathscr{D}}$, then P and P' are equivalent under the $\tilde{\mathscr{D}}$-Kripke-Kleene, $\tilde{\mathscr{D}}$-well-founded and $\tilde{\mathscr{D}}$-stable semantics.

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

The three-valued equivalence condition in this proposition is strictly stronger than the two-valued equivalence condition in Proposition 6.1. 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 $\tilde{\mathscr{D}}$-well-founded semantics and $\tilde{\mathscr{D}}$-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 $\tilde{\mathscr{D}}$.

Theorem 7.2
Let P be a stratified $\Sigma(\Pi)$-aggregate program. For any three-valued Σ-structure $\tilde{\mathscr{D}}$ approximating $\mathscr{D}$, the $\tilde{\mathscr{D}}$-well-founded model is two-valued and is equal to the standard model of P extending $\mathscr{D}$ and to the unique $\tilde{\mathscr{D}}$-stable model of P.

The proof of this result depends on the following lemma.

Lemma 7.1
Let φ be a closed $\Sigma(\Pi)$-aggregate formula such that predicates of Π do not occur in an aggregate atom. Let $\mathscr{D}$ be a Σ-structure and $\tilde{\mathscr{D}}$ be a three-valued Σ-structure

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

which is two-valued on all predicates in Σ and approximates $\mathscr{D}$. Then for any three-valued interpretation (I_1, I_2), if the predicates in Π occur only positively in φ then $\mathscr{H}_{\tilde{\mathscr{D}}(I_1,I_2)}(\varphi) = (\mathscr{H}_{\mathscr{D}(I_1)}(\varphi), \mathscr{H}_{\mathscr{D}(I_2)}(\varphi))$ and if the predicates in Π occur only negatively in φ then $\mathscr{H}_{\tilde{\mathscr{D}}(I_1,I_2)}(\varphi) = (\mathscr{H}_{\mathscr{D}(I_2)}(\varphi), \mathscr{H}_{\mathscr{D}(I_1)}(\varphi))$.

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 $p(t_1, \ldots, t_n)$:

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

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

$$\begin{aligned}\mathscr{H}_{\tilde{\mathscr{D}}(I_1,I_2)}(p(t_1,\ldots,t_n)) &= (I_1, I_2)(p([\![t_1]\!]_{\mathscr{D}}, \ldots, [\![t_n]\!]_{\mathscr{D}})) \\ &= (I_1(p([\![t_1]\!]_{\mathscr{D}}, \ldots, [\![t_n]\!]_{\mathscr{D}})), I_2(p([\![t_1]\!]_{\mathscr{D}}, \ldots, [\![t_n]\!]_{\mathscr{D}}))) \\ &= (\mathscr{H}_{\mathscr{D}(I_1)}(p(t_1,\ldots,t_n)), \mathscr{H}_{\mathscr{D}(I_2)}(p(t_1,\ldots,t_n))).\end{aligned}$$

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

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

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

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

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

$$\mathscr{H}_{\tilde{\mathscr{D}}(I_1,I_2)}(\mathrm{R}(\{\overline{x} \mid \varphi\}, t)) = \mathscr{H}_{\tilde{\mathscr{D}}}(\mathrm{R}(\{\overline{x} \mid \varphi\}, t)).$$

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

$$\mathscr{H}_{\tilde{\mathscr{D}}}(\mathrm{R}(\{\overline{x} \mid \varphi\}, t)) = (\mathscr{H}_{\mathscr{D}}(\mathrm{R}(\{\overline{x} \mid \varphi\}, t)), \mathscr{H}_{\mathscr{D}}(\mathrm{R}(\{\overline{x} \mid \varphi\}, t))).$$

□

Proof of Theorem 7.2 (Sketch)

Let P be stratified by the level mapping $\|.\|$, and let Π_i be the predicates of level i, P_i the set of rules of P with head in Π_i.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Given an interpretation $I \in \mathscr{I}$, let us define $I|_i$ as the restriction of I to the predicates of Π_i, and $I|_{\leqslant i}$ and $I|_{<i}$ as the restriction of I to the predicates of $\bigcup_{j\leqslant i}\Pi_j$, 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 $i = 1,\ldots,\|P\|$ and every $\tilde{I},\tilde{J} \in \mathscr{I}^c$, if $\tilde{I}|_{\leqslant i} = \tilde{J}|_{\leqslant i}$, then

$$\Phi^{aggr}_{P,\tilde{\mathscr{D}}}(\tilde{I})|_{\leqslant i} = \Phi^{aggr}_{P,\tilde{\mathscr{D}}}(\tilde{J})|_{\leqslant i}.$$

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

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

It follows from Corollary 3.12 in (Vennekens *et al.* 2006) that $\tilde{I}$ is the well-founded fixpoint of $\Phi^{aggr}_{P,\tilde{\mathscr{D}}}$ if and only if for every $i = 1,\ldots,\|P\|$, $\tilde{I}|_i$ is the well-founded fixpoint of $\Phi^{aggr}_{P_i,\tilde{\mathscr{D}}(\tilde{I}|_{<i})}$.

The next step is to prove by induction that for every $i = 1,\ldots,\|P\|$, the well-founded fixpoint $\tilde{I}|_i$ of $\Phi^{aggr}_{P_i,\tilde{\mathscr{D}}(\tilde{I}|_{<i})}$ is equal to $(I|_i, I|_i)$ where I is the standard model of P. This will show that the $\tilde{\mathscr{D}}$-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 $\Sigma_{i-1} = \Sigma \cup \bigcup_{j<i}\Pi_j$, $\mathscr{D}' = \mathscr{D}(I|_{<i})$ and $\tilde{\mathscr{D}}' = \tilde{\mathscr{D}}(\tilde{I}|_{<i})$. Notice that $\tilde{\mathscr{D}}'$ approximates $\mathscr{D}'$ and $\tilde{\mathscr{D}}'$ is two-valued on all predicates of Σ_{i-1} (because $\tilde{I}|_{<i}$ is a two-valued interpretation). Using Lemma 7.1 we can show that for any three-valued Π_i-interpretation (I_1, I_2),

$$\Phi^{aggr}_{P_i,\tilde{\mathscr{D}}'}(I_1, I_2) = (T^{aggr}_{P_i,\mathscr{D}'}(I^1), T^{aggr}_{P_i,\mathscr{D}'}(I^2)).$$

By Theorem 2.3 it follows that $\Phi^{aggr}_{P_i,\tilde{\mathscr{D}}'}$ is also the ultimate approximation of $T^{aggr}_{P_i,\mathscr{D}'}$. So, the $\tilde{\mathscr{D}}'$-well-founded model of P_i is equal to the ultimate well-founded model of P_i extending $\mathscr{D}'$ and, by Corollary 6.1, to the least fixpoint of $T^{aggr}_{P_i,\mathscr{D}'}$. □

Since the $\Phi^{aggr}_{P,\tilde{\mathscr{D}}}$ operators are less precise than the ultimate approximation $U^{aggr}_{P,\mathscr{D}}$ we have the following corollary of Theorems 2.1 and 7.2.

Corollary 7.1
For a stratified program P, the ultimate well-founded model is 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 $\Phi^{aggr}_{P,\tilde{\mathscr{D}}}$ 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. 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 $\mathscr{TWO}$.

Definition 7.7 (*Trivial Approximating Aggregate*)
Let $\mathrm{R} \subseteq \wp(D_1)\times D_2$ be an aggregate relation. The *trivial approximating aggregate* $triv(\mathrm{R}) : \wp(D_1)^c \times D_2 \to \mathscr{THREE}$ of R is defined as follows:

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Proposition 7.6
For every aggregate relation R, *triv*(R) is a three-valued aggregate relation and *triv*(R) approximates R.

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

As mentioned earlier, the trivial approximating aggregate is the least precise three-valued aggregate relation.

Proposition 7.7
Let R be an aggregate relation. For every three-valued aggregate relation $\mathscr{R}$ of R, *triv*(R) $\leqslant_p \mathscr{R}$.

The trivial approximating aggregates are very imprecise. Nevertheless, by Theorem 7.2, they suffice to model the semantics of the important class of stratified aggregate programs.

Corollary 7.2
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^{aggr} 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) : $\wp(D_1)^c \times D_2 \rightarrow \mathscr{THREE}$ defined as:

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

Proposition 7.8
For every aggregate relation R, *ult*(R) is a three-valued aggregate relation and *ult*(R) approximates R.

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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

Proposition 7.9
Let R be an aggregate relation. For every three-valued aggregate relation $\mathscr{R}$ which approximates R, $\mathscr{R} \leqslant_p ult(\mathrm{R})$.

So, $\Phi^{aggr}_{P,ult}$ is the most precise operator in the family of $\Phi^{aggr}_{P,\tilde{\mathscr{D}}}$ operators and by Proposition 7.1, the *ult*-well-founded and the *ult*-stable semantics are the most precise semantics of aggregate programs in this family. Of course, these semantics are still weaker than the ultimate well-founded and ultimate stable semantics from section 6. Recall that for the program $\{p \leftarrow p \vee \neg p.\}$, p is true in the ultimate well-founded model but unknown in the *ult*-well-founded model. An aggregate program showing similar behavior is given in the following example.

Example 7.3
Consider the following program with Herbrand universe $\{0, 1, 2, 3\}$:

$$p(x) \leftarrow \mathrm{CARD}_{\leqslant}(\{x|p(x)\}, 1) \vee \mathrm{CARD}_{\geqslant}(\{x|p(x)\}, 2).$$

Note that the body is a tautology in 2-valued logic. Therefore, in the ultimate semantics, this program is equivalent with:

$$p(x) \leftarrow \mathbf{t}$$

In the ultimate Kripke-Kleene, well-founded and unique stable model, p is true for all domain elements. On the other hand, in the *ult*-well-founded model, each atom $p(i)$ has value $\tilde{\mathbf{u}}$ and the program has no *ult*-stable model. This can be seen as follows. First, we observe that for each three-valued interpretation $\tilde{I}$, for every two-valued I approximated by $\tilde{I}$, $(\Phi^{aggr}_{P,ult})^2(\tilde{I})$ is an upper bound of $T^{aggr}_{P,\tilde{\mathscr{D}}}(I) = \{p(0), \ldots, p(3)\} = \top$. Hence, $(\Phi^{aggr}_{P,ult})^2$ is a constant operator and maps each three-valued interpretation $\tilde{I}$ to $\{p(0), \ldots, p(3)\}$. On the other hand, it is easy to see that for the three-valued set $\tilde{S} = \{0^{\tilde{\mathbf{u}}}, \ldots, 3^{\tilde{\mathbf{u}}}\}$, $ult(\mathrm{CARD}_{\leqslant})^1(\tilde{S}, 1)$ and $ult(\mathrm{CARD}_{\geqslant})^1(\tilde{S}, 2)$ are both false, since $\tilde{S}$ approximates sets with strictly more than one element and other sets with strictly less than two elements. Therefore,

$$(\Phi^{aggr}_{P,ult})^1(\{p(0)^{\tilde{\mathbf{u}}}, \ldots, p(3)^{\tilde{\mathbf{u}}}\}) = (\Phi^{aggr}_{P,ult})^1(\emptyset, \{p(0), \ldots, p(3)\}) = \emptyset.$$

Hence, $\emptyset$ is the least fixpoint of $(\Phi^{aggr}_{P,ult})^1(\cdot, \{p(0), \ldots, p(3)\})$ and $\{p(0), \ldots, p(3)\}$ is the least fixpoint of $(\Phi^{aggr}_{P,ult})^2(\emptyset, \cdot)$. It follows that the *ult*-well-founded fixpoint is

$$\{p(0)^{\tilde{\mathbf{u}}}, \ldots, p(3)^{\tilde{\mathbf{u}}}\} = (\emptyset, \{p(0), \ldots, p(3)\}).$$

As for the stable semantics, any *ult*-stable fixpoint is also a supported fixpoint, and the only supported fixpoint is $\{p(0), \ldots, p(3)\}$. However, the least fixpoint of $(\Phi^{aggr}_{P,ult})^1(\cdot, \{p(0), \ldots, p(3)\})$ is not $\{p(0), \ldots, p(3)\}$ but $\emptyset$. Hence, there are no *ult*-stable fixpoints.

Now we look at characterizations of the ultimate approximating aggregate of some common aggregate functions. Such characterizations are useful for several

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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

R	$((S_1, S_2), d) \in ult(\text{R})^1$ iff	$((S_1, S_2), d) \in ult(\text{R})^2$ iff
MIN	$d \in S_1 \wedge \text{MIN}(S_2, d)$	$d \in S_2 \wedge \neg\exists d' \in S_1 : d' < d$
MAX	$d \in S_1 \wedge \text{MAX}(S_2, d)$	$d \in S_2 \wedge \neg\exists d' \in S_1 : d' > d$
GLB	$\text{GLB}(S_1, d) \wedge \text{GLB}(S_2, d)$	$\text{GLB}(S_1 \cup (S_2 \cap [d, \top]), d)$
LUB	$\text{LUB}(S_1, d) \wedge \text{LUB}(S_2, d)$	$\text{LUB}(S_1 \cup (S_2 \cap [\bot, d]), d)$

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 7.10
Let $\text{R} : \wp(D_1) \times D_2$ be a monotone aggregate relation. Then $((S_1, S_2), d) \in ult(\text{R})^1$ if and only if $(S_1, d) \in \text{R}$ and $((S_1, S_2), d) \in ult(\text{R})^2$ if and only if $(S_2, d) \in \text{R}$.

Proposition 7.11
Let $\text{R} : \wp(D_1) \times D_2$ be an anti-monotone aggregate relation. Then $((S_1, S_2), d) \in ult(\text{R})^1$ if and only if $(S_2, d) \in \text{R}$ and $((S_1, S_2), d) \in ult(\text{R})^2$ if and only if $(S_1, d) \in \text{R}$.

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

Proposition 7.12
The characterizations in Table 2 are correct.

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

Proposition 7.13 ($ult(\text{CARD})$)
For every three-valued set (S_1, S_2) and element d:

$$((S_1, S_2), d) \in ult(\text{CARD})^1 \text{ if and only if } |S_1| = d = |S_2|$$
$$((S_1, S_2), d) \in ult(\text{CARD})^2 \text{ if and only if } |S_1| \leqslant d \leqslant |S_2|$$

Proposition 7.14 ($ult(\text{SUM})$ *and* $ult(\text{PROD})$)
For every three-valued set (S_1, S_2) and element d:

$$((S_1, S_2), d) \in ult(\text{SUM})^1 \text{ iff } \text{SUM}(S_1, d) \wedge \forall (x_1, \ldots, x_n) \in S_2 \setminus S_1 : x_1 = 0$$
$$((S_1, S_2), d) \in ult(\text{SUM})^2 \text{ iff } \exists S' \subseteq S_2 \setminus S_1 : \text{SUM}(S_1 \cup S', d)$$

$$((S_1, S_2), d) \in ult(\text{PROD})^1 \text{ iff } \text{PROD}(S_1, d) \wedge (d = 0 \vee \forall (x_1, \ldots, x_n) \in S_2 \setminus S_1 : x_1 = 1)$$
$$((S_1, S_2), d) \in ult(\text{PROD})^2 \text{ iff } \exists S' \subseteq S_2 \setminus S_1 : \text{PROD}(S_1 \cup S', d).$$

Proof
The proof is straightforward and uses only basic properties of numbers. □

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Note that the definition of $ult(\text{SUM})^2$ and $ult(\text{PROD})^2$ is simply a reformulation of the original definition of *ult*. In fact, Pelov (2004) shows that the complexity of computing $ult(\text{SUM})^2$ and $ult(\text{PROD})^2$ is NP-complete so it is unlikely that any efficient algorithms can be found.

Finally, we look at combined aggregate relations of the form $\text{F}_{\geqslant}$ and $\text{F}_{\leqslant}$ where $\text{F} : \wp_F(D_1) \to D_2$ is an aggregate function on finite sets and $\leqslant$ is a total order on D_2. For all three aggregate functions CARD, SUM, and PROD we can give efficient algorithms for $ult(\text{F}_{\geqslant})^1$ and $ult(\text{F}_{\geqslant})^2$. We start with the following general result. Let $\min_{\text{F}}, \max_{\text{F}} : \wp_F(D_1)^c \to \wp_F(D_1)$ 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 $\text{F}(S)$ is minimal (resp. maximal) over the set $[S_1, S_2]$, i.e.,

$$\forall S' \in [S_1, S_2] : \text{F}(\min_{\text{F}}(S_1, S_2)) \leqslant \text{F}(S')$$
$$\forall S' \in [S_1, S_2] : \text{F}(\max_{\text{F}}(S_1, S_2)) \geqslant \text{F}(S').$$

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 $ult(\text{F}_{\leqslant})$ and $ult(\text{F}_{\geqslant})$ can be computed using the $\min_{\text{F}}$ and $\max_{\text{F}}$ functions in the following way.

Proposition 7.15
Let $\text{F} : \wp_F(D_1) \to D_2$ be an aggregate function and $\leqslant$ be a total order on D_2. Then:

$$((S_1, S_2), d) \in ult(\text{F}_{\geqslant})^1 \text{ if and only if } \text{F}(\min_{\text{F}}(S_1, S_2)) \geqslant d$$
$$((S_1, S_2), d) \in ult(\text{F}_{\geqslant})^2 \text{ if and only if } \text{F}(\max_{\text{F}}(S_1, S_2)) \geqslant d$$

and similarly for $ult(\text{F}_{\leqslant})$:

$$((S_1, S_2), d) \in ult(\text{F}_{\leqslant})^1 \text{ if and only if } \text{F}(\max_{\text{F}}(S_1, S_2)) \leqslant d$$
$$((S_1, S_2), d) \in ult(\text{F}_{\leqslant})^2 \text{ if and only if } \text{F}(\min_{\text{F}}(S_1, S_2)) \leqslant d.$$

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

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

The proofs of the other cases are analogous. □

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Proposition 7.16
If F is a monotone aggregate function with respect to $\leqslant$ 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 $\leqslant$ 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 in 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 $\{(x_1, \ldots, x_n) \in S \mid x_1 \geqslant 0\}$ and S^- denotes the set $\{(x_1, \ldots, x_n) \in S \mid x_1 < 0\}$.

Proposition 7.17
For every three-valued set (S_1, S_2):

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

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 7.17. Below, $S^{[1,\infty)}$ denotes the set $\{(x_1, \ldots, x_n) \in S \mid x_1 \in [1, \infty)\}$ and $S^{[0,1)}$ denotes the set $\{(x_1, \ldots, x_n) \in S \mid x_1 \in [0, 1)\}$.

Proposition 7.18
For the aggregate function $\text{PROD}^{\mathbb{R}^+} : \wp_F(\mathbb{R}^+) \to \mathbb{R}^+$,

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

The algorithms for PROD on the entire set of real numbers are more complicated and can be found in (Pelov 2004).

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

Example 7.4 (*Shortest Path*)
Consider the following formulation of the problem of finding the shortest path (Van Gelder 1992, Example 4.1:

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 7.19
Let $edge^{\mathscr{D}}$ 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 7.4 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: acyclic finite graphs, finite graphs with non-negative weights and infinite graphs with weights in $\mathbb{N}_0$. It follows from Proposition 7.19 that for these types of graphs, the programs in Example 5.4 and Example 7.4 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 true $sp(a,b,w)$ 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 $T^{aggr}_{P,\mathscr{D}}$ which we defined in section 5.1. The key to the proof of this result is the following lemma.

Lemma 7.2
Let $\tilde{\mathscr{D}} = ult(\mathscr{D})$ and let (I_1,I_2) be a three-valued interpretation. If φ is a closed positive aggregate formula then $\mathscr{H}_{\tilde{\mathscr{D}}(I_1,I_2)}(\varphi) = (\mathscr{H}_{\mathscr{D}(I_1)}(\varphi), \mathscr{H}_{\mathscr{D}(I_2)}(\varphi))$. If φ is a closed negative aggregate formula then $\mathscr{H}_{\tilde{\mathscr{D}}(I_1,I_2)}(\varphi) = (\mathscr{H}_{\mathscr{D}(I_2)}(\varphi), \mathscr{H}_{\mathscr{D}(I_1)}(\varphi))$.

Proof
The proof extends that of Lemma 7.1 with several new cases when φ is an aggregate atom $\mathrm{R}(\{\overline{x} \mid \psi\}, t)$, ψ contains defined predicates and $\mathrm{R}^{\mathscr{D}}$ is either a monotone or an anti-monotone aggregate relation. We consider only the case when $\mathrm{R}^{\mathscr{D}}$ is a monotone

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

aggregate relation and ψ is a positive aggregate formula. The other three cases ($\mathrm{R}^{\mathscr{D}}$ monotone and ψ negative and $\mathrm{R}^{\mathscr{D}}$ anti-monotone and ψ positive or negative) are symmetric. First, note that since $\psi(\overline{x})$ is a positive formula then for every tuple of domain elements $\overline{d}$, $\psi(\overline{d})$ is also positive. So,

$$\mathscr{H}_{(I_1,I_2)}(\psi(\overline{d})) = (\mathscr{H}_{I_1}(\psi(\overline{d})), \mathscr{H}_{I_2}(\psi(\overline{d})))$$

and consequently

$$[\![\{\overline{x} \mid \psi\}]\!]_{\tilde{\mathscr{D}}(I_1,I_2)} = ([\![\{\overline{x} \mid \psi\}]\!]_{\mathscr{D}(I_1)}, [\![\{\overline{x} \mid \psi\}]\!]_{\mathscr{D}(I_2)}).$$

We have:

$$\begin{aligned}
\mathscr{H}_{\tilde{\mathscr{D}}(I_1,I_2)}(\mathrm{R}(\{\overline{x} \mid \psi\}, t)) &= ult(\mathrm{R})([\![\{\overline{x} \mid \psi\}]\!]_{\tilde{\mathscr{D}}(I_1,I_2)}, [\![t]\!]) \\
&= ult(\mathrm{R})(([\![\{\overline{x} \mid \psi\}]\!]_{\mathscr{D}(I_1)}, [\![\{\overline{x} \mid \psi\}]\!]_{\mathscr{D}(I_2)}), [\![t]\!]) \\
\text{(by Proposition 7.10)} \quad &= (\mathrm{R}([\![\{\overline{x} \mid \psi\}]\!]_{\mathscr{D}(I_1)}, [\![t]\!]), \mathrm{R}([\![\{\overline{x} \mid \psi\}]\!]_{\mathscr{D}(I_2)}, [\![t]\!])) \\
&= (\mathscr{H}_{\mathscr{D}(I_1)}(\mathrm{R}(\{\overline{x} \mid \psi\}, t)), \mathscr{H}_{\mathscr{D}(I_2)}(\mathrm{R}(\{\overline{x} \mid \psi\}, t))).
\end{aligned}$$

□

Theorem 7.3
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 $M = \mathrm{lfp}(T^{aggr}_{P,\mathscr{D}})$.

Proof
From Lemma 7.2 follows that if P is a positive aggregate program then

$$\Phi^{aggr}_{P,ult}(I_1, I_2) = (T^{aggr}_{P,\mathscr{D}}(I_1), T^{aggr}_{P,\mathscr{D}}(I_2)).$$

By Theorem 2.3 follows that $\Phi^{aggr}_{P,ult}$ is also the ultimate approximation of $T^{aggr}_{P,\mathscr{D}}$. So, the *ult*-well-founded model of P is equal to the ultimate well-founded model of P extending $\mathscr{D}$ and, by Corollary 6.1, to the least fixpoint of $T^{aggr}_{P,\mathscr{D}}$. □

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 $ult(\mathrm{SUM})^2$ and $ult(\mathrm{PROD})^2$ is NP-complete (Pelov 2004) 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 $\mathrm{F} : \wp_F(D_1) \to D_2$ be an aggregate function and $\langle D_2, \leqslant \rangle$ be a totally ordered set. The *bound approximating aggregate* $bnd(\mathrm{F}) : \wp_F(D_1)^c \times D_2 \to \mathscr{THREE}$ is defined as follows

$$((S_1, S_2), d) \in bnd(\mathrm{F})^1 \text{ if and only if } \mathrm{F}(\min_{\mathrm{F}}(S_1, S_2)) = d = \mathrm{F}(\max_{\mathrm{F}}(S_1, S_2))$$
$$((S_1, S_2), d) \in bnd(\mathrm{F})^2 \text{ if and only if } \mathrm{F}(\min_{\mathrm{F}}(S_1, S_2)) \leqslant d \leqslant \mathrm{F}(\max_{\mathrm{F}}(S_1, S_2)).$$

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Note that by using Proposition 7.15, the definition of $bnd(\mathrm{F})$ is equivalent to $ult(\mathrm{F}_{\geqslant}) \wedge ult(\mathrm{F}_{\leqslant})$ where $\wedge$ is the greatest lower bound in $\mathscr{THREE}$ with respect to the $\leqslant$ order (see Example 2.1).

Proposition 7.20
Let $\mathrm{F}: \wp_F(D_1) \to D_2$ be an aggregate function and $\langle D_2, \leqslant \rangle$ be a totally ordered set. Then $bnd(\mathrm{F})$ is a three-valued aggregate relation of F.

It is interesting to see how $bnd(\mathrm{F})$ compares to $ult(\mathrm{F})$. We first show that the first components of the two three-valued aggregate relations are always equal.

Proposition 7.21
Let $\mathrm{F}: \wp_F(D_1) \to D_2$ be an aggregate function and $\leqslant$ a total order on D_2. Then $bnd(\mathrm{F})^1 = ult(\mathrm{F})^1$.

However, for most aggregate functions, $bnd(\mathrm{F})$ can be less precise than $ult(\mathrm{F})$, i.e., $bnd(\mathrm{F}) <_p ult(\mathrm{F})$ because $bnd(\mathrm{F})^2 > ult(\mathrm{F})^2$.

Example 7.5
The pair $((\emptyset, \{1,3\}), 2)$ is not in the relation $ult(\mathrm{SUM})^2$ because there is no set $S \in [\emptyset, \{1,3\}]$ such that $\mathrm{SUM}(S) = 2$. On the other hand $((\emptyset, \{1,3\}), 2) \in bnd(\mathrm{SUM})^2$ because

$$b_1 = \mathrm{SUM}(\min_{\mathrm{SUM}}(\emptyset, \{1,3\})) = \mathrm{SUM}(\emptyset) = 0,$$
$$b_2 = \mathrm{SUM}(\max_{\mathrm{SUM}}(\emptyset, \{1,3\})) = \mathrm{SUM}(\{1,3\}) = 4,$$

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

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 $\tilde{\mathscr{D}}$ 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 $Aggr(\Sigma)$ and for varying but finite input $\Sigma \setminus Aggr(\Sigma)$-structures $\mathscr{D}_o$. The problem with this intuition is that the interpretation $\mathrm{R}^{\tilde{\mathscr{D}}}$ of a given aggregate symbol R in this class is of course not really fixed: it varies with the input structure $\mathscr{D}_o$. 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 $\mathscr{D}_o$.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

To circumvent this problem, we introduce the following concepts. Let us fix an aggregate program P based on signature $\Sigma(\Pi)$. Consider the class $\mathscr{C}$ of two-valued structures of $\Sigma \setminus Aggr(\Sigma)$ with a finite domain (i.e., with finite domains for every sort s). We assume a given function X from $\mathscr{C}$ to the class of Σ-structures such that for each $\mathscr{D}_o \in \mathscr{C}$, $X(\mathscr{D}_o)$ extends $\mathscr{D}_o$ with three-valued aggregates $\mathscr{R}$ for each aggregate symbol $\mathrm{R} \in Aggr(\Sigma)$. Moreover, we assume that for each aggregate symbol $\mathrm{R} \in Aggr(\Sigma)$, there are two Turing machines which, for (an appropriate encoding of) arbitrary $\mathscr{D}_o \in \mathscr{C}$ and arbitrary well-typed pair (S, d) consisting of a three-valued set $\tilde{S}$ and a domain element d, compute respectively $(\mathrm{R}^{X(\mathscr{D}_o)})^1(\tilde{S}, d)$ and $(\mathrm{R}^{X(\mathscr{D}_o)})^2(\tilde{S}, d)$.

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

Theorem 7.4

- deciding if an atom A is true in the Kripke-Kleene model of a program P extending a structure in the class $X(\mathscr{C})$ is in P;
- deciding if an atom A is true in the well-founded model of a program P extending a structure in the class $X(\mathscr{C})$ is in P;
- deciding the existence of an exact stable model of a program P extending a structure in the class $X(\mathscr{C})$ 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 $s_1 \times \cdots \times s_n$). Then, since the number of elements in such a set is bounded by a polynomial in the size of the domain of $\mathscr{D}_o$, computing the value of the aggregate is certainly polynomial in the size of the domain of the input structure $\mathscr{D}_o$.

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 $\Sigma \setminus Aggr(\Sigma)$-structure $\mathscr{D}_o$, the lattice in which the computations take place is the power-set lattice $\mathscr{I} = \wp(base_{\mathscr{D}_o}(\Pi))$. The maximal chain length in this lattice is the number of facts, i.e., the cardinality of $base_{\mathscr{D}_o}(\Pi)$. This number is polynomial in the size of the domain of $\mathscr{D}_o$. It follows then that all we need to prove to obtain the desired complexity results is that for any given pair $\tilde{J} \in \mathscr{I}$, we can compute $\Phi^{aggr}_{P,X(\mathscr{D}_o)}(\tilde{J})$ in polynomial time in the size of the domain.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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

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 $\mathrm{R}(s, d)$ 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 $(\mathscr{R}^{X(\mathscr{D}_o)})^1(\tilde{S}, d)$ or $(\mathscr{R}^{X(\mathscr{D}_o)})^2(\tilde{S}, d)$. It follows straightforwardly from the induction hypothesis that the first task can be done in polynomial time in the domain size of $\mathscr{D}_o$, whereas the second task is polynomial by assumption. □

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.

7.5 *Summary of the results*

In this and the previous section, we have introduced a family of Kripke-Kleene, a family of stable and a family of well-founded semantics for aggregate programs, parameterized by the approximation operator. We introduced also a sub-family of these semantics, obtained by extending the standard three-valued Fitting operator with different three-valued aggregates. We presented two generic ways for deriving a three-valued aggregate from a given (two-valued) aggregate relation, called the *trivial* and the *ultimate* three-valued aggregate.

All instances in each of the three families are *consistent* with each other. In particular, when one instance of the Kripke-Kleene or well-founded semantics infers that a literal is true, there is no instance in which this literal is false and any more precise instance of the same semantics will infer the same literal. Also, a model in one instance of the stable semantics is also a model in each more precise instance

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

of the stable semantics. This shows that all instances of each of the three families of semantics basically formalize the same intuitions but with different degree of precision, with the semantics based on the ultimate operator as the most precise. However, for several important subclasses of aggregate programs, optimal precision is reached using weaker approximations. As a general rule, when the well-founded model of one approximation is two-valued, then it coincides with the well-founded and unique stable model of each more precise approximation. In Theorem 7.3 and Corollary 6.1, we proved that for definite aggregate programs, the least fixpoint semantics, the *ult*-well-founded, *ult*-stable, the ultimate well-founded and the ultimate stable semantics all coincide. One example is the Company Control Example 5.2. For stratified aggregate programs, Theorem 7.2 showed that the standard model, the *triv*-well-founded and *triv*-stable model and the ultimate well-founded and stable model all coincide. This class was illustrated by the first Shortest Path Example 7.4. In case of the second Shortest Path Example 7.4, Proposition 7.19 showed that (under some weak conditions) the *ult*-well-founded model is two-valued.

This raises the question of what a good choice of the semantics is and how we can exploit our results to build an effective reasoning system for aggregate programs. Just as for standard logic programming, the family of semantics of aggregate programs offers a trade-off between *precision* versus *complexity*. Therefore, choosing the "right" semantics is a *pragmatic* matter and the relevant question is what degree of precision is required for the applications that one has in mind.

In the case of logic programs without aggregates, the standard and the ultimate semantics only differ for programs with a rare combination of reasoning by cases and recursion, as in $\{p \leftarrow p \vee \neg p\}$. We are not aware of a single non-artificial example of a logic program, appearing in the literature, which shows this behaviour. This means that, de facto, the standard and the ultimate semantics coincide and the ultimate semantics can be computed using the more efficient Fitting operator. In case of aggregate programs, the situation seems even better. It must be admitted that applications of recursion over aggregates in the literature are quite rare. We believe that this is not a coincidence. Indeed, aggregates provide a rather powerful way to avoid recursion. In a language without aggregates, the means to compute the value of some aggregate of a set of objects (e.g., cardinality, sum, minimum or maximum,...) is often by using recursion over the potential elements of the set (Van Gelder 1992). By allowing aggregates in the language, such applications of recursion can be avoided. Recall that for stratified aggregates programs even the *triv*-stable and *triv*-well-founded semantics are sufficiently precise. As for the applications of recursion over aggregates in this paper, the Party Invitation program of Example 5.1 is to be interpreted under the supported model semantics, whereas for the second Shortest Path Example 7.4 and the Company Control Example 5.2, the *ult*-well-founded model is the correct solution.

In summary, we believe that extensions of the standard semantics with polynomially computable three-valued aggregates, provide sufficient precision for almost all practical applications. This is the case for the ultimate approximating aggregates of CARD, MIN, MAX and the bound approximating aggregates of SUM and PROD (Pelov 2004). To us, it seems that the most suitable approximation for being incorporated

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

in model generation or query systems for aggregate programs, is the extension of the Fitting operator with those three-valued aggregates.

8 Related work

Aggregate relations are closely related to generalized quantifiers (Lindström 1966). An example of a generalized quantifier is $\textsc{most}(A, B)$, defined on page 309, 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 formalized similarly. For example, the cardinality aggregate could be formalized 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 *super-valuation 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 $I \models body(r)$ and $I \subseteq J$ implies $J \models body(r)$ 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

$$\exists z \mathrm{R}(\{\overline{x} \mid q(\overline{x})\}, z) \wedge p(z, t) \tag{1}$$

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

$$\mathrm{R}_P(\{\overline{x} \mid q(\overline{x})\}, t) \tag{2}$$

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).

A common approach to extend the stable semantics of general logic programs with negation to aggregate programs was to treat aggregate literals as negative

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

literals when computing the program reduct (Kemp and Stuckey 1991; Gelfond 2002; Elkabani *et al.* 2004). Such semantics is relatively easy to define and the definition also extends to answer sets of disjunctive logic programs. However, it has been shown that this semantics accepts non-minimal models and does not model correctly some problems with recursion over aggregation, for example the Company Controls program from Example 5.2 Pelov (2004, Section 4.3.3). For aggregate programs containing only negative aggregate literals it has been shown (Pelov *et al.* 2004) that the set of *ult*-stables models is the same as the set of stable models defined by the above authors.

A more elaborate definition of a stable semantics was given by (Simons *et al.* 2002) for programs with weight constraints and implemented by the well-known smodels system. In our language, weight constraints correspond to aggregate atoms build with the $\text{SUM}_{\leqslant}$ and $\text{SUM}_{\geqslant}$ aggregate relations. An extensive comparison between the *ult*-stable semantics and the stable semantics of weight constraints can be found in Pelov *et al.* (2004); Pelov (2004) and will not be repeated here.

A novel feature of the language of weight constraints was that it allows weight constraints to be present also in the head of the rules. This approach have been further developed in different directions. One line of research was to consider different variations and extensions of weight constraints like abstract constraints (Marek and Truszczyński 2004), monotone cardinality atoms (Marek *et al.* 2004) or set constraints (Marek and Remmel 2004). Such constraint atoms correspond in a natural way to aggregate atoms. The stable semantics of these extensions is also defined in terms of lattice operators. However, since constraint atoms are allowed in the heads of rules, the operators become non-deterministic and the algebraic theory is quite different than the approximation theory we used in this work. However, all the semantics agree on the class of definite aggregate programs and its least model semantics. The equivalent of a definite logic program in Marek and Remmel (2004) is called a Horn SC-logic programs and such programs are also characterized by a unique model which is the least fixpoint of a deterministic monotone operator S_P which is the equivalent of our $T^{aggr}_{P,\mathscr{D}}$ operator.

Another extension of the language of weight constraint atoms is to considers arbitrary propositional formulas containing arbitrary aggregate atoms (both in the head and in the body of the rule) (Ferraris 2005). The answer set semantics for such propositional formulas is different than the $\tilde{\mathscr{D}}$-stable semantics which we defined. The main reason is that the semantics of Ferraris (2005) is based on the logic of here-and-there (Lifschitz *et al.* 2001) while the $\tilde{\mathscr{D}}$-stable semantics are based on Kleene's strong three-valued logic $\mathscr{THREE}$. The simplest example of the difference between the two logics is that $\neg\neg p$ is equivalent to p in $\mathscr{THREE}$ while it is not in the logic of here-and-there. As a consequence, the program consisting of the single rule $p \leftarrow \neg\neg p$ is a definite program according to Definition 5.6 because $\neg\neg p$ is a positive (aggregate) formula. So, all our semantics assign to this program the model $\emptyset$ On the other hand, it has two models $\emptyset$ and $\{p\}$ according to the answer set semantics of (Lifschitz *et al.* 2001; Ferraris 2005). This difference also manifests for aggregate formulas. For example, it is easy to see that the aggregate formulas $\neg\text{SUM}_{\neq}(s,t)$ and $\text{SUM}(s,t)$ have the same three-valued truth value for any interpretation of SUM with

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

a three-valued aggregate relation while they are not equivalent under the semantics of (Ferraris 2005).

Another proposal for a stable semantics of disjunctive logic programs extended with aggregates was given in Faber *et al.* (2004). In the sequel we investigate in more detail the relationship with this semantics to the family of $\tilde{\mathscr{D}}$-stable semantics defined earlier. First, we recall the definitions of the stable semantics of Faber *et al.* (2004).

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 $inst(P)$ 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.

In one direction we can show the following result.

Proposition 8.1
For any aggregate program P and for any three-valued structure $\tilde{\mathscr{D}}$, if an interpretation I is an $\tilde{\mathscr{D}}$-stable model of P then I is a FLP-stable model of P.

Proof
We will show that any $\tilde{\mathscr{D}}$-stable model I is also a $\tilde{\mathscr{D}}$-stable model of P^I. It will then follow from Lemma 2.1, that I is a minimal pre-fixpoint of $T^{aggr}_{P^I,\mathscr{D}}$. Since the pre-fixpoints of $T^{aggr}_{P^I,\mathscr{D}}$ are exactly the models of P^I, this will imply that I is a minimal model of P^I and hence, I is a FLP-stable model of P. We start by showing

$$\forall J \subseteq I : (\Phi^{aggr}_{P^I,\tilde{\mathscr{D}}})^1(J,I) = (\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^1(J,I). \qquad (*)$$

This will imply that $\mathrm{lfp}((\Phi^{aggr}_{P^I,\tilde{\mathscr{D}}})^1(\cdot,I)) = \mathrm{lfp}((\Phi^{aggr}_{P,\tilde{\mathscr{D}}})^1(\cdot,I)) = I$, 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 $\mathscr{H}^1_{\tilde{\mathscr{D}}(J,I)}(body(r)) = \mathbf{f}$. The definition of reduct implies that for such a rule $\mathscr{H}_{\mathscr{D}(I)}(body(r)) = \mathbf{f}$ where $\mathscr{D}$ is the (two-valued) structure approximated by $\tilde{\mathscr{D}}$. We also have that $(J,I) \leqslant_p (I,I)$ which implies that

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

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

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

Example 8.1
Consider the following aggregate program P:

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

The program 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

$$(\neg p(A) \wedge \neg p(B)) \vee (p(A) \wedge p(B)). \tag{3}$$

In fact, if we interpret $\text{CARD}_{\neq}$ 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 identical[3]. It follows that the program P and the program obtained by substituting the rules

$$r \leftarrow \neg p(A) \wedge \neg p(B).$$

$$r \leftarrow p(A) \wedge p(B).$$

for the first rule in P, have identical three-valued immediate consequence operators and hence, have identical stable models[4]. 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.

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 natural principle is not satisfied by the semantics defined in Faber *et al.* (2004). This is a fundamental weakness of this semantics.

Finally, we mention another recent work on defining a stable semantics of aggregates (Son and Pontelli 2007). As shown in that paper, the semantics defined there coincides with the *ult*-stable semantics and provides an interesting different formalization for this semantics.

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

[3] The three-valued equivalence of $ult(\text{CARD}_{\neq})$ and (3) is an application of a general transformation of ultimate approximations of aggregate atoms to formulas (Pelov 2004, Section 5.3.6.)

[4] 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.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 2004).

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 $\tilde{\mathscr{D}}$-well-founded model which is equal to the unique $\tilde{\mathscr{D}}$-stable model of P and the standard model of P for any three-valued interpretation $\tilde{\mathscr{D}}$ 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 models.

Appendix A Proof of the shortest path theorem

Before proving Proposition 7.19, we first introduce some concepts.

We define the *length* of a path as its number of edges. We define a partial function $sp(.,.)$ as follows: $sp(a,b)$ is defined if and only if there is a shortest path from a to b and $sp(a,b) = w$ where w is the weight of the shortest path. 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 7.19
Let $edge^{\mathscr{D}}$ 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 7.4 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:

- $I_0 = \bot$;
- $J_n = St^{\uparrow}_{\Phi_P}(I_n) = \mathrm{lfp}(\Phi^2_P(I_n,\cdot))$;
- $I_n = St^{\downarrow}_{\Phi_P}(J_{n-1}) = \mathrm{lfp}(\Phi^1_P(\cdot,J_{n-1}))$.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

We now construct this sequence until we reach a fixpoint.

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

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

In the first iteration $\Phi_P^2(\bot, \bot) = T_{P,\mathscr{D}}^{aggr}(\bot) = C_1$ where

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

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:

$$[\![\{c \mid cp(a,b,c)\}]\!]_{(\bot,C_1)} = (\emptyset, W_{ab})$$

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

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

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

$$\Phi_P^2(\bot, \cdot) \uparrow (2n-1) = \bigcup_{1 \leqslant i \leqslant n} C_i \cup \bigcup_{1 \leqslant i < n} S_i$$
$$\Phi_P^2(\bot, \cdot) \uparrow (2n) = \bigcup_{1 \leqslant i \leqslant n} C_i \cup \bigcup_{1 \leqslant i \leqslant n} S_i$$

where

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

At the first limit ordinal ω we have

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

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 $I_1 = \mathrm{lfp}(\Phi_P^1(\cdot, C \cup S)) = CE \cup SP$ where:

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

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

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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

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

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

$$[\![\{c \mid cp(a,b,c)\}]\!]_{(\bot, C \cup S)} = (\emptyset, A_{ab})$$

where

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

Further, $ult(\text{MIN})^1((\emptyset, A_{ab}), w)$ 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

$$[\![\{c \mid cp(a,b,c)\}]\!]_{(CE_1, C \cup S)} = (W_{ab}, A_{ab})$$

and $((W_{ab}, A_{ab}), w) \in ult(Min)^1$ 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 $edge^{\mathscr{D}}$ 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

$$[\![\{c \mid cp(a,b,c)\}]\!]_{(CE \cup SP, CE \cup SP)} = (B_{ab}, B_{ab})$$

where

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

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

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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 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:

- $I_0 = \bot$.
- $J_0 = \{sp(a, a, -n-1), sp(a, b, -n), cp(a, a, -n-1), cp(a, b, -n) \mid n \in \mathbb{N}\}$. This describes all paths in the graph.
- $I_1 = CE \cup SP = \{cp(a, a, -1), cp(a, b, 0)\}$. Indeed, since there are no shortest paths in this graph, the set SP is empty and CE is just a copy of the edge relation.
- $J_1 = \text{lfp}(\Phi_P^2(CE, \cdot))$. 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 $\bot$. It holds that

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

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.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

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.

Davey, B. A. and Pristley, H. A. 1990. *Introduction to Lattices and Order*. Cambridge University Press.

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 *18th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 847–852.

Denecker, M. 2000. Extending classical logic with inductive definitions. In *1st International Conference on Computational Logic*. 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 Truszczyński, M. 2004. Ultimate approximations and its application in nonmonotonic knowledge representation. *Information and Computation 192,* 1, 84–121.

Denecker, M., Pelov, N. and Bruynooghe, M. 2001. Ultimate well-founded and stable model semantics for logic programs with aggregates. In *17th International Conference on Logic Programming*. LNCS, vol. 2237. Springer, 212–226.

Elkabani, I., Pontelli, E. and Son, T. C. 2004. Smodels with CLP and its applications: A simple and effective approach to aggregates in ASP. In *International Conference on Logic Programming*. 73–89.

Faber, W., Leone, N. and Pfeifer, G. 2004. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *9th European Conference on Artificial Intelligence (JELIA)*. LNCS, vol. 3229. Springer, 200–212.

Ferraris, P. 2005. Answer sets for propositional theories. In *8th International Conference on Logic Programming and Nonmonotonic Reasoning*. LNCS, vol. 3662. Springer, 119–131.

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. LNCS, 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.* MIT Press, 1070–1080.

Kemp, D. B. and Stuckey, P. J. 1991. Semantics of logic programs with aggregates. In *International Logic Programming Symposium.* MIT Press, 387–401.

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

Lindström, P. 1966. First order predicate logic with generalized quantifiers. *Theoria 32*, 186–195.

Marek, V., Niemelä, I. and Truszczyński, M. 2004. Logic programs with monotone cardinality atoms. In *7th International Conference on Logic Programming and Nonmonotonic Reasoning*. LNCS, vol. 2923. Springer, 155–166.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press

Marek, V. and Remmel, J. 2004. Set constraints in logic programming. In *7th International Conference on Logic Programming and Nonmonotonic Reasoning*. LNCS, vol. 2923. Springer, 167–179.

Marek, V. W. and Truszczyński, M. 2004. Logic programs with abstract constraint atoms. In *National Conference on Artificial Intelligence*. AAAI Press / The MIT Press, 86–91.

Mitchell, D. and Ternovska, E. 2005. A framework for representing and solving NP-search problems. In *Proc. of the National Conference on Artificial Intelligence*. 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*. Morgan Kaufmann, 264–277.

Pelov, N. 2004. 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*. 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.

Son, T. C. and Pontelli, E. 2007. A constructive semantic characterization of aggregates in answer set programming. *Theory and Practice of Logic Programming*. Accepted as a Technical Note.

van Eijck, J. 1996. Quantifiers and partiallity. 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*.

Vennekens, J., Gilis, D. and Denecker, M. 2006. Splitting an operator: Algrebraic modularity results for logics with fixpoint semantics. *ACM Transactions on Computational Logic*. Accepted.

https://doi.org/10.1017/S1471068406002973 Published online by Cambridge University Press