Aggregation in Disjunctive Deductive Databases

Dietmar Seipel
University of Würzburg, Dept. of Computer Science
Am Hubland, D – 97074 Würzburg, Germany
seipel@informatik.uni-wuerzburg.de
Tel. +49 931 888 5026, Fax. +49 931 888 4600

Abstract

Practical applications of deductive databases often require the use of extended features like *disjunctive information*, *aggregation operators* or *default negation*. But it has been unclear how one could deal with aggregation in the presense of *recursion* and *disjunction*. Usually, there is the requirement that deductive databases must be *stratified w.r.t. aggregation*, i.e. a predicate q that is defined using aggregation based on some other predicate p may not be used for deriving this other predicate p.

Disjunctive deductive databases extend the classical DATALOG language by disjunction, default negation, and function symbols.[1] There are practical examples that require non–stratified aggregation in disjunctive deductive databases.

We will present a clear *declarative set semantics* for disjunctive deductive databases $\mathcal{P}$ with aggregation. The semantics is based on a special *program transformation*, which replaces an aggregation by a suitable construct using *default negation* and the function symbol for lists. The transformed database $\mathcal{P}_{agg}$ is free of aggregation, and it can be evaluated based on the well–known concept of *stable models*.

Keywords

disjunctive logic programming, aggregation, stratification, non–monotonic reasoning, stable models

[1] In this paper we will use the term disjunctive deductive database as an equivalent to disjunctive logic program.

1 Introduction

Many practical applications of relational or deductive databases require the use of *aggregation operators*. For instance a study of *data mining* and *decision support* applications ([15, 9]) shows that aggregation is used heavily – often even *user–defined* aggregates are required – and moreover aggregation is frequently used in forms that cannot be supported well or efficiently by the current SQL–standard ([15]). In SQL3, aggregates and default negation can only be used in stratified queries, and in relational or deductive databases so far there is no support for uncertain, disjunctive information.

Typical and well–known forms of aggregation are the following: temporal aggregation in spatio–temporal databases ([8, 14]), aggregation in online–analytic processing (OLAP, [9]), computing a bill–of–materials, finding the shortest path in a graph, market analysis for transitive ownership and control of corporations for finding trusts ([1, 15]).

In the DISLOG–project ([11, 12]) a system for reasoning with disjunctive information in deductive databases has been developed. DISLOG has been used for *diagnostic reasoning*, reasoning in *spatial databases*, or for solving *configuration* problems. So far, aggregation had not been considered in disjunctive deductive databases.

1.1 An Example with Non–Stratified Aggregation

The following example of packing a knapsack requires disjunctive deductive databases with non–stratified aggregation. There are similar configuration examples in other domains.

Example 1.1 (Packing a Knapsack)

The problem is to pack a knapsack with items from a relation

$$item(Item, Weight),$$

such that the following conditions are satisfied:

(i) the total weight of the knapsack does not exceed a limit given by

$$weight\_limit(Limit_W);$$

(ii) if the total weight '*Total_Weight*' of the knapsack exceeds a limit given by

$$energy\_limit(Limit_E),$$

then the knapsack must contain an '*Energy_Drink*';

(iii) if '*Total_Weight*' does not exceed this limit, then the knapsack may not contain an '*Energy_Drink*';

(iv) the knapsack must contain either an '*Umbrella*' or a '*Trenchcoat*', or both.

The problem can be solved using the following disjunctive deductive database $\mathcal{P}_{kn}$ with aggregation. An atom $knapsack(Item, Weight)$ indicates that the knapsack contains the item '*Item*' with the corresponding weight '*Weight*'. The total weight of the knapsack is computed by the aggregation rule r_0. In $\mathcal{P}_{kn}$ disjunction is used in rule r_4 for formulating the choice (iv) on the fact level. Furthermore, the integrity constraints given in (i) and (iii) are formulated as disjunctive rules r_1 and r_3 with zero head atoms.

$$\begin{aligned} r_0 &= knapsack\_weight(< Weight >_{sum}) \leftarrow \\ &\quad knapsack(Item, Weight) \end{aligned}$$

$$\begin{aligned} r_1 &= \leftarrow weight\_limit(Limit_W) \wedge \\ &\quad knapsack\_weight(Total\_Weight) \wedge \\ &\quad Total\_Weight > Limit_W \end{aligned}$$

$$\begin{aligned} r_2 &= knapsack('Energy\_Drink', Weight) \leftarrow \\ &\quad item('Energy\_Drink', Weight) \wedge \\ &\quad energy\_limit(Limit_E) \wedge \\ &\quad knapsack\_weight(Total\_Weight) \wedge \\ &\quad Total\_Weight > Limit_E \end{aligned}$$

$$\begin{aligned} r_3 &= \leftarrow knapsack('Energy\_Drink', Weight) \wedge \\ &\quad energy\_limit(Limit_E) \wedge \\ &\quad knapsack\_weight(Total\_Weight) \wedge \\ &\quad Total\_Weight \leq Limit_E \end{aligned}$$

$$\begin{aligned} r_4 &= knapsack('Umbrella', Weight_1) \vee \\ &\quad knapsack('Trenchcoat', Weight_2) \leftarrow \\ &\quad item('Umbrella', Weight_1) \wedge \\ &\quad item('Trenchcoat', Weight_2) \end{aligned}$$

1.2 The Transformation Approach

In our new approach, a disjunctive deductive database $\mathcal{P}$ with aggregation will be evaluated by transforming it into another disjunctive deductive database $\mathcal{P}_{agg}$ without aggregation, which then can be evaluated using the well–known semantics of stable models.

Even though the database $\mathcal{P}$ can be disjunctive, aggregation in $\mathcal{P}$ will be treated in a *definite manner* based on the individual stable models of $\mathcal{P}_{agg}$. This means that the aggregation condition will hold within each of the stable models (which can be considered as ordinary relational databases, possibly with function symbols).

The aggregation operator *agg* can be defined by arbitrary (disjunctive) rules for a corresponding binary predicate $\oplus_{agg}$, which are added to $\mathcal{P}_{agg}$: a call '$\oplus_{agg}(Vs, V)$' must compute the value V of the aggregation operator *agg* on the set corresponding to the list *Vs*. Thus, in our approach arbitrary *user–defined* aggregation operators are possible.

The transformation adds a binary function symbol for lists to $\mathcal{P}$. If $\mathcal{P}$ did not contain any function symbols, then $\mathcal{P}_{agg}$ only contains this restricted use of the list function symbol, which is used for collecting the values to be aggregated.

If the original database $\mathcal{P}$ was *stratified* w.r.t. negation and aggregation, then $\mathcal{P}_{agg}$ is stratified w.r.t. negation, too. Then the semantics of *perfect models* can be used.[2]

[2]Perfect models coincide with stable models on databases that are stratified w.r.t. negation, but they are conceptually simpler and they can be computed more efficiently.

The rest of the paper is organized as follows: In Section 2 we present the syntax of disjunctive deductive databases with aggregation, and their predicate dependency graph. In Section 3 we recapitulate the well–known stable model semantics of disjunctive deductive databases (without aggregation). In Section 4 the declarative semantics of disjunctive deductive databases with aggregation is defined based on a special program transformation. In Section 5 we give some concluding remarks.

2 Disjunctive Deductive Databases with Aggregation

Given a first order language $\mathcal{L}$, a *disjunctive deductive database* $\mathcal{P}$ (cf. [5]) consists of logical inference rules of the form

$$\begin{aligned} r \;\; = \;\; & A_1 \vee \ldots \vee A_k \leftarrow \\ & B_1 \wedge \ldots \wedge B_m \wedge \\ & not\, C_1 \wedge \ldots \wedge not\, C_n, \end{aligned} \qquad (1)$$

where A_i, $i \in \langle 1,k \rangle$, B_i, $i \in \langle 1,m \rangle$, and C_i, $i \in \langle 1,n \rangle$, are atoms in the language $\mathcal{L}$, $k,m,n \in I\!N_0$, and *not* is the negation–by–default operator.[3] A rule is called a fact if $m = n = 0$. The set of all *ground instances* of the rules and facts in $\mathcal{P}$ is denoted by *gnd* $(\mathcal{P})$. A rule (or database) is called *positive–disjunctive* if it does not contain default negation (i.e. $n = 0$). It is called *normal* if $k = 1$, and *definite* if $k = 1$ and $n = 0$. A rule r of the form (1) is denoted for short as $r = \alpha \leftarrow \beta \wedge not \cdot \gamma$, where $\alpha = A_1 \vee \ldots \vee A_k$, $\beta = B_1 \wedge \ldots \wedge B_m$, and $\gamma = C_1 \vee \ldots \vee C_n$.[4] α is called the *head*, β is called the *positive body*, and $not \cdot \gamma$ is called the *negative body* of r.

Aggregation

A disjunctive rule r with *aggregation* is given by a rule of the form (1) above, such that the head atoms $A_i = a_i(X_i^1, \ldots, X_i^{n_i})$ may have *aggregation arguments* X_i^j of the form

$$X_i^j = <V>_{agg},$$

where '*agg*' can be one of the common aggregation operators '*sum*', '*max*', '*min*', '*count*', '*avg*' (average), '*set*' (cf. [8]), and 'V' is a variable symbol that also occurs in some positive body atom B_j. A disjunctive deductive database $\mathcal{P}$ with *aggregation* is given by a set of disjunctive rules with aggregation. In our approach even *user–defined* aggregation operators are possible, which can be defined by arbitrary (disjunctive) rules.

Note that we allow for aggregation arguments only in rule heads. This is not really a restriction, since an aggregation in the body of a rule can be simulated by introducing a new rule – with this aggregation in the head – defining a new predicate, which is then used in the body of the original rule.

For example, the following rule joins two relations p and a, groups the result by X, and then computes the sum of the different V–values

$$r = q(X, <V>_{sum}) \leftarrow p(X,Y,V) \wedge a(Y).$$

The following SQL–statement corresponds to the rule r:

```
INSERT INTO q
SELECT p.attr(1), sum(p.attr(3))
FROM p, a
WHERE p.attr(2) = a.attr(1)
GROUP BY p.attr(1)
```

When we compare the *expressive power* of disjunctive rules with aggregation and of SQL–statements with aggregation, then the question arises, whether we can also represent a HAVING–clause in the form of a logical inference rule. Roughly speaking, this can be done by first defining a new predicate without using the selection condition of the HAVING–clause and by then applying the selection condition to the resulting predicate.

The Predicate Dependency Graph

The *predicate dependency graph* $G_{\mathcal{P}} = \langle N_{\mathcal{P}}, E_{\mathcal{P}} \rangle$ of a disjunctive deductive database $\mathcal{P}$ with aggregation is given by a set $N_{\mathcal{P}} = \cup_{r \in \mathcal{P}} N_r$ of nodes and a set $E_{\mathcal{P}} = \cup_{r \in \mathcal{P}} E_r$ of labeled edges, where each rule $r \in \mathcal{P}$ results in a set of nodes and edges. For a rule r of the

[3] By $I\!N_+$ we denote the set $\{1,2,3,\ldots\}$ of positive natural numbers, whereas $I\!N_0$ denotes the set $\{0,1,2,\ldots\}$ of all natural numbers. $\langle n,m \rangle$ denotes the interval $\{n, n+1, \ldots, m\}$ of natural numbers.

[4] Note that γ is a disjunction, and, according to De Morgan's law, $not \cdot \gamma$ is taken to be a conjunction.

form (1) above, let the predicate symbols in the atoms $A_i = a_i(\overline{X_i})$, $B_j = b_j(\overline{Y_j})$, $C_l = c_l(\overline{X_l})$ be a_i, b_j, c_l, respectively. Then

$$\begin{aligned} N_r &= \{\, a_i \mid i \in \langle 1, k \rangle \,\} \cup \\ &\quad \{\, b_j \mid j \in \langle 1, m \rangle \,\} \cup \{\, c_l \mid l \in \langle 1, n \rangle \,\}, \\ E_r &= \{\, (a_i \xrightarrow{+} a_j) \mid i, j \in \langle 1, k \rangle, i \neq j \,\} \cup \\ &\quad \{\, (a_i \xrightarrow{+} b_j) \mid i \in \langle 1, k \rangle, j \in \langle 1, m \rangle, \\ &\qquad A_i \text{ contains no aggregation } \} \cup \\ &\quad \{\, (a_i \xrightarrow{agg} b_j) \mid i \in \langle 1, k \rangle,\ j \in \langle 1, m \rangle, \\ &\qquad A_i \text{ contains an aggregation } < V >_{agg} \} \cup \\ &\quad \{\, (a_i \xrightarrow{-} c_j) \mid i \in \langle 1, k \rangle, j \in \langle 1, n \rangle \,\}. \end{aligned}$$

A *cycle* for a predicate symbol p in $G_{\mathcal{P}}$ is given by a sequence

$$\mathcal{C}_p = ((p_i \xrightarrow{l_i} p_{i+1}))_{1 \leq i \leq h}$$

of labeled edges, such that $p = p_1 = p_{h+1}$.

(i) $\mathcal{P}$ is called *stratified w.r.t. negation* (or just *stratified*), if there exists no cycle $\mathcal{C}_p$ in $G_{\mathcal{P}}$, such that one of the edges $(p_i \xrightarrow{l_i} p_{i+1})$ is labeled by '$l_i = -$'.

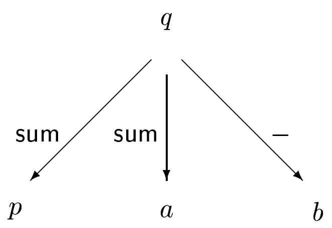
(ii) $\mathcal{P}$ is called *stratified w.r.t. aggregation*, if there exists no cycle $\mathcal{C}_p$ in $G_{\mathcal{P}}$, such that one of the edges $(p_i \xrightarrow{l_i} p_{i+1})$ is labeled by an aggregation operator '$l_i = agg$'.

For example, both of the following disjunctive deductive databases are stratified w.r.t. negation:

(i) The disjunctive deductive database $\mathcal{P} = \{\, r \,\}$ consisting of the rule

$$\begin{aligned} r \ = \ & q(X, < V >_{sum}) \leftarrow \\ & \quad p(X, Y, V) \wedge a(Y) \wedge \\ & \quad not\, b(X, V) \end{aligned}$$

is also stratified w.r.t. aggregation, and its predicate dependency graph is given by:



(ii) The knapsack packing example $\mathcal{P}_{kn}$ of the introduction is non–stratified w.r.t. aggregation, since there exits the following cycle through aggregation:

$$(knapsack\_weight \xrightarrow{sum} knapsack),$$
$$(knapsack \xrightarrow{+} knapsack\_weight).$$

3 Stable Models for Disjunctive Deductive Databases

The Herbrand base $HB_{\mathcal{P}}$ of a disjunctive deductive database $\mathcal{P}$ contains all ground atoms over the language of $\mathcal{P}$. An *Herbrand interpretation* of $\mathcal{P}$ is given by a subset $I \subseteq HB_{\mathcal{P}}$. It defines a mapping I: $HB_{\mathcal{P}} \to \{\, \mathsf{t}, \mathsf{f} \,\}$ that assigns a truth value "t" (*true*) or "f" (*false*) to each atom in $HB_{\mathcal{P}}$: $I(A) = \mathsf{t} \Leftrightarrow A \in I$. [5] I is an *Herbrand model* of a ground rule $r = \alpha \leftarrow \beta \wedge not \cdot \gamma$ if

$$I(\beta) \wedge \neg I(\gamma) = \mathsf{t} \ \Rightarrow \ I(\alpha) = \mathsf{t}.$$

I is an *Herbrand model* of $\mathcal{P}$ if I is an Herbrand model of all ground instances $r \in gnd(\mathcal{P})$ of the elements of $\mathcal{P}$. I is a *minimal model* of $\mathcal{P}$ if I is an Herbrand model of $\mathcal{P}$ and there is no other Herbrand model J of $\mathcal{P}$ such that $J \subsetneq I$. The set of all *minimal models* of $\mathcal{P}$ is denoted by $\mathcal{MM}(\mathcal{P})$.

For positive–disjunctive deductive databases $\mathcal{P}$, the semantics is given by $\mathcal{MM}(\mathcal{P})$, and a formula ϕ is considered to be a *consequence* of $\mathcal{P}$ under the minimal model semantics if ϕ holds in all minimal models of $\mathcal{P}$.

The declarative semantics of disjunctive deductive databases in general is defined based on a more general class of Herbrand models, the *stable models*, which are defined in terms of the minimal models of the *Gelfond–Lifschitz transformation* (GL–transformation, [2, 6]) of $\mathcal{P}$. The GL–transformation of $\mathcal{P}$ w.r.t. an Herbrand interpretation M is obtained from the ground instance *gnd* ($\mathcal{P}$) of $\mathcal{P}$ by replacing in every rule the negative body by its truth value:[6]

[5] For $A_i \in HB_{\mathcal{P}}$, $i \in \langle 1, k \rangle$, and a connective $\theta \in \{\, \vee, \wedge \,\}$ we define $I(\, A_1 \theta \ldots \theta A_k \,) = I(A_1) \theta \ldots \theta I(A_k)$. For $k = 0$, the empty disjunction (i.e. $\theta = \vee$) evaluates to f, whereas the empty conjunction (i.e. $\theta = \wedge$) evaluates to t.

[6] It holds $M(not \cdot \gamma) = \neg M(\gamma)$. If this truth value is "t", then "t" can be deleted from the body. If it is "f", then the whole rule can be deleted from $\mathcal{P}^M$.

(i) For $r = \alpha \leftarrow \beta \wedge not{\cdot}\gamma \in gnd\,(\mathcal{P})$ we define $r^M = \alpha \leftarrow \beta \wedge \neg M(\gamma)$.

(ii) The GL–transformation of $\mathcal{P}$ w.r.t. M is $\mathcal{P}^M = \{\, r^M \mid r \in gnd\,(\mathcal{P}) \,\}$.

$\mathcal{P}^M$ is a ground positive–disjunctive database with the truth values t and f as additional atoms,[7] where rules with f in the body can be deleted from $\mathcal{P}^M$.

Definition 3.1 (Stable Models, [2, 6])
Let M be a Herbrand interpretation of a disjunctive deductive database $\mathcal{P}$.

(i) M is called *stable model* of $\mathcal{P}$ if $M \in \mathcal{MM}(\mathcal{P}^M)$.

(ii) The set of all stable models of $\mathcal{P}$ is denoted by $\mathcal{STAB}(\mathcal{P})$.

Stable models are also minimal models, i.e. $\mathcal{STAB}(\mathcal{P}) \subseteq \mathcal{MM}(\mathcal{P})$, but the converse in general is not true.

A formula ϕ is considered to be a *consequence* of a disjunctive deductive database $\mathcal{P}$ under the stable model semantics if ϕ holds in all stable models of $\mathcal{P}$.

Example 3.2
For the disjunctive deductive database $\mathcal{P}_{win}$ consisting of the following rules and facts

$$\begin{aligned} & win(X) \leftarrow move(X,Y) \wedge not\; win(Y), \\ & move(a,b) \vee move(a,c), \\ & move(b,d),\; move(c,d), \end{aligned}$$

we get a set with two stable models, which coincide on the subset I of atoms:

$$\begin{aligned} \mathcal{STAB}(\mathcal{P}_{win}) \;=& \\ & \{\, \{\, move(a,b) \,\} \cup I, \\ & \;\; \{\, move(a,c) \,\} \cup I \,\}, \\ I \;=&\; \{\, move(b,d),\; move(c,d), \\ & \quad win(b),\; win(c) \,\}. \end{aligned}$$

For $M = \{\, move(a,b) \,\} \cup I$ we get that $M(not\; win(y)) = \mathsf{t}$ iff $y \in \{a,d\}$, and thus we get the following GL–transformation $\mathcal{P}^M_{win}$:

$$\begin{aligned} \mathcal{P}^M_{win} \;=\; & \{\; win(x) \leftarrow move(x,y) \mid \\ & \quad x \in \{a,b,c,d\},\; y \in \{a,d\} \;\} \cup \\ & \{\; move(a,b) \vee move(a,c), \\ & \quad move(b,d),\; move(c,d) \;\}. \end{aligned}$$

Obviously it holds $M \in \mathcal{MM}(\mathcal{P}^M_{win})$.

All of the atoms in I and the disjunction $move(a,b) \vee move(a,c)$ are consequences of $\mathcal{P}_{win}$ under the stable model semantics.

[7] Note that these truth values must evaluate to themselves under all Herbrand interpretations I of $\mathcal{P}^M$.

4 The Declarative Semantics for Aggregation

We will present a clear *declarative semantics* for disjunctive deductive databases $\mathcal{P}$ with aggregation. The semantics is based on a special program transformation, which replaces all aggregations of $\mathcal{P}$ by suitable constructs using *default negation*. The transformed database $\mathcal{P}_{agg}$ is free of aggregation, and it can be evaluated based on the concept of stable models.

In a *preprocessing step* all rules $r = \alpha \leftarrow \beta \wedge not{\cdot}\gamma$ are normalized by replacing the complete rule body $\beta \wedge not \cdot \gamma$ by a single atom $B_r = body_r(X_1, \ldots, X_h)$, which has a new predicate symbol $body_r$ and has as arguments all variable symbols $X_1, \ldots, X_h$ that occur in $\beta \wedge not \cdot \gamma$. Thus, we can replace r by the following two rules r_1 and r_2:

$$\begin{aligned} r_1 &= B_r \leftarrow \beta \wedge not \cdot \gamma, \\ r_2 &= \alpha \leftarrow B_r. \end{aligned}$$

Only the rule r_2 can contain aggregation. It will be treated like described below, where we assume that the bodies of the rules with aggregation consist of exactly one atom.

Transformation of Definite Rules

Consider the following prototypical disjunctive rule r with aggregation, which computes the aggregation of V–values of a predicate 'p' after grouping by X–values. If a V–value occurs with several (different) Y–values, then it contributes several times to the aggregation.

$$r \;=\; q(X, < V >_{agg}) \leftarrow\; p(X,Y,V)$$

Note that 'X', 'Y' and 'V' could also be vectors of argument positions, and correspondingly 'agg' could be a vector of aggregation functions for the components of 'V'.

The rule r is replaced by the following normal rules, which are free of aggregation:

$$\begin{array}{rcl} r^*_{agg} & = & p^*_{agg}(X, [(Y,V)|\,YVs], [V|\,Vs]) \leftarrow \\ & & \quad p^*_{agg}(X, YVs, Vs) \wedge \\ & & \quad p(X,Y,V) \wedge \\ & & \quad not\ member((Y,V), YVs) \\ e^*_{agg} & = & p^*_{agg}(X, [(Y,V)], [V]) \leftarrow \\ & & \quad p(X,Y,V) \\ r_{agg} & = & p_{agg}(X,V) \leftarrow \\ & & \quad p^*_{agg}(X, YVs, Vs) \wedge \\ & & \quad not\ p^{ext}_{agg}(X, YVs) \wedge \\ & & \quad \oplus_{agg}(Vs, V) \\ e_{agg} & = & p^{ext}_{agg}(X, YVs) \leftarrow \\ & & \quad p(X,Y,V) \wedge \\ & & \quad not\ member((Y,V), YVs) \\ r' & = & q(X,V) \leftarrow \\ & & \quad p_{agg}(X,V) \end{array}$$

The *transformed database* is given by

$$\mathcal{P}_{agg} = (\,\mathcal{P} \setminus \{\, r \,\}\,) \cup \mathcal{P}^r_{agg},$$

where

$$\mathcal{P}^r_{agg} = \{\, r^*_{agg},\, e^*_{agg},\, r_{agg},\, e_{agg},\, r' \,\}.$$

Instead of aggregation, $\mathcal{P}_{agg}$ contains rules with PROLOG–style function symbols '[ ]' for lists. We also need the usual definition of the predicate '$member$'. The predicate 'p^*_{agg}' serves for constructing lists *'YVs'*, which consist of pairs '(Y,V)' such that '$p(X,Y,V)$' holds, and for constructing the corresponding lists *'Vs'* of the values 'V'. The call '$p^{ext}_{agg}(X, YVs)$' in r_{agg} checks whether the list *'YVs'* can be extended by other pairs '(Y,V)' that are not yet contained in it. If it cannot be extended, then the call '$\oplus_{agg}(Vs, V)$' applies the aggregation function 'agg' to the list *'Vs'* and produces the final result 'V'.[8]

Obviously, in our approach the aggregation function 'agg' can be defined by arbitrary (disjunctive) rules for $\oplus_{agg}$. We only have to require that the same result $p_{agg}(X,V)$ is derived by the call $\oplus_{agg}(Vs,V)$ in r_{agg} from $p^*_{agg}(X, YVs, Vs)$ for all permutations YVs.

In an *implementation* of the list approach the inefficiency problem arising from permutations can be avoided by simply using *ordered lists* or *ordered sets*, depending on whether we are preferring a bag semantics or a set semantics. In the declarative description of our set semantics, however, we have not addressed this problem.

The *predicate dependency graph* $G_{\mathcal{P}^r_{agg}}$ is given in Figure 1, where labels "+" are omitted. Instead of the edge $(q \xrightarrow{sum} p)$ due to r in $G_{\mathcal{P}}$, the graph $G_{\mathcal{P}_{agg}}$ contains a path with a negative edge:

$$(q \xrightarrow{+} p_{agg}),\ (p_{agg} \xrightarrow{-} p^{ext}_{agg}),\ (p^{ext}_{agg} \xrightarrow{+} p).$$

I.e., the aggregation has been *transformed* into a negation. If the original database $\mathcal{P}$ was *stratified w.r.t. negation and aggregation*, then $\mathcal{P}_{agg}$ is *stratified w.r.t. negation*, too.

Transformation of Disjunctive Rules

Disjunctive rules r with more than one head atom, some of which contain aggregation, can also be treated by this replacement technique:

$$\begin{array}{rcl} r & = & q_1(X_1, <V_1>_{agg_1}) \vee \ldots \vee \\ & & \quad q_{k_1}(X_{k_1}, <V_{k_1}>_{agg_{k_1}}) \vee \\ & & \quad A_1 \vee \ldots \vee A_{k_2} \leftarrow \\ & & \quad p(X_1, \ldots, X_{k_1}, Y, V_1, \ldots, V_{k_1}) \end{array}$$

All the definite rules

$$\begin{array}{rcl} r_i & = & q_i(X_i, <V_i>_{agg_i}) \leftarrow \\ & & \quad p(X_1, \ldots, X_{k_1}, Y, V_1, \ldots, V_{k_1}) \end{array}$$

are transformed as described above into rules $r^*_{agg_i}, e^*_{agg_i}, r_{agg_i}, e_{agg_i}$, which define predicates $p_{agg_i}(X_i, V_i)$. Then the new rule is

$$\begin{array}{rcl} r' & = & q_1(X_1, V_1') \vee \ldots \vee q_{k_1}(X_{k_1}, V_{k_1}') \vee \\ & & \quad A_1 \vee \ldots \vee A_{k_2} \leftarrow \\ & & \quad p_{agg_1}(X_1, V_1') \wedge \ldots \wedge \\ & & \quad p_{agg_{k_1}}(X_{k_1}, V_{k_1}') \wedge \\ & & \quad p(X_1, \ldots, X_{k_1}, Y, V_1, \ldots, V_{k_1}), \end{array}$$

with k_1 pairwise different, fresh variable symbols V_i'. Note that the atom with the predicate symbol p is necessary in the body of r',

[8] If 'agg' is based on an *associative* binary operator, which holds for 'sum', 'max', 'min', '$count$', then the aggregation could of course be evaluated incrementally without first constructing the whole list *'Vs'*.

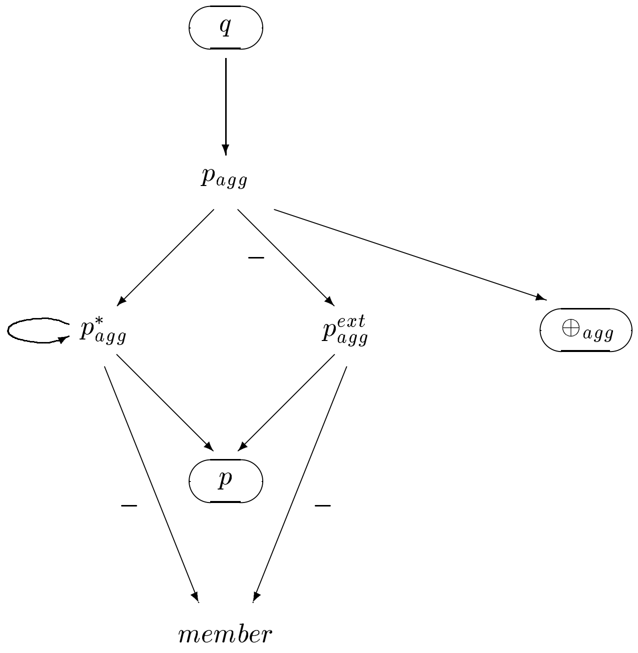


Figure 1: The Predicate Dependency Graph of $\mathcal{P}^r_{agg}$

since it can share arguments with the atoms in $A_1 \vee \ldots \vee A_{k_2}$.

The Stable Model Semantics with Aggregation

The declarative semantics of a disjunctive deductive database $\mathcal{P}$ with aggregation is defined based on its transformed database $\mathcal{P}_{agg}$ using the concept of stable models:

$$\mathcal{STAB}_{agg}(\mathcal{P}) = \{\, I \cap H\!B_{\mathcal{P}} \mid I \in \mathcal{STAB}(\mathcal{P}_{agg}) \,\}.$$

For each stable model of $\mathcal{P}_{agg}$ only the *regular* atoms $A \in I \cap H\!B_{\mathcal{P}}$ are considered; all *artificial* atoms with predicate symbols that were introduced during the program transformation (i.e. p^*_{agg}, p_{agg}, p^{ext}_{agg}, *member*) are thrown away.

The Knapsack Example

The aggregation rule $r = q(X, {<}V{>}_{agg}) \leftarrow p(X,Y,V)$ of $\mathcal{P}_{kn}$ is definite. It is

$$r_0 = \mathit{knapsack\_weight}({<}\,\mathit{Weight}\,{>}_{sum}) \leftarrow \mathit{knapsack}(\mathit{Item}, \mathit{Weight}).$$

The predicate on which we have to aggregate is '$p = \mathit{knapsack}$', the resulting predicate is '$q = \mathit{knapsack\_weight}$', the argument vector '$X$' is empty, '$Y = \mathit{Item}$', and '$V = \mathit{Weight}$'.

$\mathcal{P}_{kn}$ is non–stratified w.r.t. aggregation, since '*knapsack_weight*' is used for defining '*knapsack*'. Thus, $(\mathcal{P}_{kn})_{agg}$ is non–stratified w.r.t. negation, and it must be evaluated by stable model semantics (rather than perfect model semantics, which would be simpler). Given the basic facts in

$$\begin{aligned} I = \{\; & \mathit{weight\_limit}(10),\ \mathit{energy\_limit}(6), \\ & \mathit{item}(\mathit{'Umbrella'}, 2), \\ & \mathit{item}(\mathit{'Trenchcoat'}, 1), \\ & \mathit{item}(\mathit{'Energy\_Drink'}, 2), \\ & \mathit{item}(\mathit{'Mineral\_Water'}, 3), \\ & \mathit{item}(\mathit{'Bread'}, 2) \;\}, \end{aligned}$$

and the rule

$$\begin{aligned} r_5 = \; & \mathit{knapsack}(\mathit{Item}, \mathit{Weight}) \vee \\ & \mathit{leave\_at\_home}(\mathit{Item}, \mathit{Weight}) \leftarrow \\ & \mathit{item}(\mathit{Item}, \mathit{Weight}) \wedge \\ & \mathit{Item} \neq \mathit{'Umbrella'} \wedge \end{aligned}$$

$$Item \neq \text{'}Trenchcoat\text{'} \wedge$$
$$Item \neq \text{'}Energy\_Drink\text{'},$$

in principle all items fit into the knapsack. There exist 10 allowed packings, i.e. stable models of $(\mathcal{P}_{kn})_{agg}$, out of 32 possible combinations of the 5 items.[9]

5 Conclusions

Currently a number of prototypical deductive database systems are available, e.g. LDL++, CORAL, Aditi, Glue–NAIL! ([13, 8]), which basically allow for stratified aggregation. E.g. LDL++ supports aggregates with choice in XY–stratified programs.

Our approach extends deductive database technology by allowing for disjunction and aggregation; even *user–defined* aggregation operators are possible. It is not necessary that the aggregation operator *agg* is associative or that it allows for an incremental evaluation, which is necessary for the approach of [15] for deductive databases with aggregation. We are currently implementing the new approach in DisLog using an *ordered sets* representation for the values that have to be aggregated.

In the special case of DATALOG with associative, stratified aggregation, our semantics coincides with the traditional semantics that is used by current deductive database systems.

References

[1] *S. Ceri, G. Gottlob, L. Tanca:* Logic Programming and Databases, Springer, 1990.

[2] *M. Gelfond, V. Lifschitz:* The Stable Model Semantics for Logic Programming, Proc. Fifth Intl. Conference and Symposium on Logic Programming 1988 (ICSLP'88), MIT Press, 1988, pp. 1070–1080.

[3] *F. Giannotti, D. Pedreschi, C. Zaniolo:* Semantics and Expressive Power of Non–Deterministic Constructs in Deductive Databases, to appear in JCCS.

[9] All allowed knapsacks contain exactly one of the items *'Umbrella'* and *'Trenchcoat'*. There are 4 packings where the knapsack does not contain *'Mineral_Water'* (in this case it cannot contain an *'Energy_Drink'* and *'Bread'* is optional) and 6 packings where it contains *'Mineral_Water'* (in this case an *'Energy_Drink'* is contained in three packings).

[4] *J.W. Lloyd:* Foundations of Logic Programming, second edition, Springer, 1987.

[5] *J. Lobo, J. Minker, A. Rajasekar:* Foundations of Disjunctive Logic Programming, MIT Press, 1992.

[6] *T.C. Przymusinski*: Stable Semantics for Disjunctive Programs, New Generation Computing, vol. 9, 1991, pp. 401–424.

[7] *W.G. Roth, R. Ramakrishnan*: MIMSY: A System for Analyzing Time Series in the Stock Market Domain, Proc. Workshop on Applications of Deductive Systems at the Intl. Symposium on Logic Programming 1993 (ILPS'93).

[8] *R. Ramakrishnan, D. Srivastava, S. Sudarshan*: CORAL – Control, Relations and Logic, Proc. Intl. Conference on Very Large Databases 1992 (VLDB'92).

[9] *R. Ramakrishnan:* Database Management Systems, McGraw–Hill, 1998.

[10] *D. Seipel, J. Minker, C. Ruiz*: Model Generation and State Generation for Disjunctive Logic Programs, Journal of Logic Programming, vol. 32(1), 1997, pp. 48–69.

[11] *D. Seipel, H. Thöne*: DisLog – A System for Reasoning in Disjunctive Deductive Databases, Proc. Intl. Workshop on the Deductive Approach to Information Systems and Databases 1994 (DAISD'94), pp. 325–343.

[12] *D. Seipel:* DisLog – A Disjunctive Deductive Database Prototype, Proc. Twelfth Workshop on Logic Programming (WLP'97), 1997, pp. 136–143.
DisLog is available on the WWW at "http://www-info1.informatik.uni-wuerzburg.de/databases/DisLog".

[13] *C. Zaniolo, N. Arni, K. Ong*: Negation and Aggregates in Recursive Rules: the LDL++ Approach, Proc. Intl. Conf. on Deductive and Object–Oriented Database systems 1993 (DOOD'93), Springer, 1993, pp. 204–221.

[14] *C. Zaniolo, S. Ceri, C. Faloutsos, R.T. Snodgrass, V.S. Subrahmanian, R. Zicari:* Advanced Database Systems, Morgan Kaufmann, 1997.

[15] *C. Zaniolo, H. Wang*: Logic–Based User–Defined Aggregates for the Next Generation of Database Systems, in: The Logic Programming Paradigm: Current Trends and Future Directions, K.R. Apt, V.W. Marek, M. Truszczyński, D.S. Warren (eds.), Springer, 1999.