Probabilistic Datalog: Implementing Logical Information Retrieval for Advanced Applications

Norbert Fuhr
University of Dortmund

Abstract

In the logical approach to information retrieval (IR), retrieval is considered as uncertain inference. Whereas classical IR models are based on propositional logic, we combine Datalog (function-free Horn clause predicate logic) with probability theory. Therefore, probabilistic weights may be attached to both facts and rules. The underlying semantics extends the well-founded semantics of modularly stratified Datalog to a possible worlds semantics. By using default independence assumptions with explicit specification of disjoint events, the inference process always yields point probabilities. We describe an evaluation method and present an implementation. This approach allows for easy formulation of specific retrieval models for arbitrary applications, and classical probabilistic IR models can be implemented by specifying the appropriate rules. In comparison to other approaches, the possibility of recursive rules allows for more powerful inferences, and predicate logic gives the expressiveness required for multimedia retrieval. Furthermore, probabilistic Datalog can be used as a query language for integrated information retrieval and database systems.

1 Introduction

The logical view on information retrieval (IR) treats retrieval as inference. For a query q, the system is searching for documents d that imply the query logically, i.e. for which the logical formula $q \leftarrow d$ is true. Due to the intrinsic vagueness and imprecision of IR, a logic that allows for uncertain reasoning should be used. In [Rijsbergen 86], a probabilistic approach is discussed for this purpose, thus document retrieval can be based on the estimation of the probability $P(q \leftarrow d)$. This approach sets the framework for probabilistic IR methods.

Numerous evaluations have demonstrated the feasibility of probabilistic models for text retrieval, even for large-scale databases (see e.g. [Harman 95], [Crestani et al. 98]). The probabilistic parameters involved in these models can be estimated either from relevance feedback data, or by combining collection statistics with appropriate assumptions.

In [Wong & Yao 95], it is shown that all classical IR models (probabilistic as well as Boolean, fuzzy and vector space models) can interpreted as probabilistic inference. All these models are based on propositional logic (in combination with probability theory), where e.g. terms are treated as propositions. However, for new IR applications (e.g. structured documents, hypertext, multimedia objects), the expressiveness of propositional logic is not sufficient. In order to represent the temporal or spatial relationships within multimedia objects or for modelling complex terminologies, some kind of predicate logic is required. However, a full-fledged theorem prover for first order logic (in combination with probabilistic reasoning) would not be appropriate for dealing with the large amounts of data an IR system typically has to cope with. For this reason, we are looking for a compromise between efficiency and expressiveness.

In the database field, there are similar problems. In fact, the logical view on databases interpretes retrieval as the task of finding all objects o from the database that imply the query q, i.e. make the implication $q \leftarrow o$ true (see e.g. [Ullman 88]). The logical view on IR can be seen as a generalization of this approach, by

switching from certain to uncertain inference. For deductive databases, a variant of Horn predicate logic called Datalog is widely used. Regarding IR as generalization of database retrieval, it seems quite natural to develop a probabilistic version of Datalog. However, as described in [Pearl 88, pp. 4–12], the combination of a rule-based approach with probability theory leads to certain restrictions and inconsistencies. This is due to the fact that the former is based on extensional semantics, whereas the latter requires intensional semantics in order to yield correct results.

In this paper, we present a new approach for combining Datalog with probability theory that overcomes these difficulties. This is achieved by using intensional semantics in combination with logical rules. As a result, we present a new version of probabilistic Datalog (pD).

The major advantages of pD are the following:

- The rule-based approach allows for easy formulation of retrieval models for specific or novel applications, like e.g. combination with a thesaurus or retrieval in hypertext bases or hierarchically structured documents.
- Classical probabilistic IR models can be formulated in pD by appropriate rules, since they are just special cases of a more general probabilistic inference mechanism.
- Since pD allows for recursive rules, it provides more powerful inference than any other (implemented) probabilistic IR model.
- Using Horn clause predicate logic as basis yields the expressiveness required for dealing with new kinds of IR applications.
- Finally, since pD is a generalization of (deterministic) Datalog, it can be used as a standard query language for both IR and database systems, and thus also for integration of these two types of systems on the logical level.

The remainder of this paper ist structured as follows: First, we give an informal introduction into pD. Section 3 describes syntax and semantics and section 4 the evaluation of pD. In section 5, we introduce a restricted version of pD that uses default independence assumptions and explicit disjointness of events for computing point probabilities. Then we give further application examples, compare our work with similar approaches, and finally give some conclusions and an outlook on future work.

2 Informal description of pD

Probabilistic Datalog is an extension of Datalog with negation (see e.g. [Ullman 88], [Ceri et al. 90]). On the syntactical level, the major difference is that with ground facts and rules, also a probabilistic weight may be given, e.g.

`0.7 indterm(d1,ir).  0.8 indterm(d1,db).`

Informally speaking, the probabilistic weight gives the probability that the following fact or rule is true. In our example, the index term 'ir' is assigned to document d1 with a probabilistic weight of 0.7 and the term 'db' (databases) with a weight of 0.8. Retrieving documents dealing with both of these topics now can be accomplished by means of the rule (as usual in Prolog or Datalog, variables start with capital letters and constants with lower-case letters)

`q1(X)` ← `indterm(X,ir) , indterm (X,db).`

Obviously, document d1 fulfills predicate q1 with a certain probability. Let us assume that index terms are stochastically independent. Then we can compute a probability of 0.56 for the probabilistic AND-combination in this example. In a similar way, the OR-combination produced by the rules

`q2(X)` ← `indterm(X,ir).  q2(X)` ← `indterm(X,db).`

would give us probability 0.94 for `q2(d1)`.

As a more interesting example, we can use pD rules for performing retrieval in hypertext structures where we have directed links between single documents (or nodes). For example, the fact `link(d1,d2)` denotes that there is a link from `d1` to `d2`. Assuming that two documents linked together are also semantically related, we can formulate a probabilistic rule for a predicate `related`:

```
link(d2,d1).  link(d3,d2).
0.5 related(D1,D2) ← link(D1,D2).
```
Now we state that a document is about a term when it is directly indexed with this term or when it is related to another document that is about the same term:
```
about(D,T) ← indterm(D,T).
about(D,T) ← related(D,D1) , about(D1,T).
```
Note that due to the recursive definition, a document also may be about a term if it is only indirectly related to another document indexed with this term. Thus, the query
```
?- about(X,db).
```
now would return three documents, namely d1 with probability 0.8, d2 with probability $0.5 \cdot 0.8 = 0.4$ and d3 with probability $0.5 \cdot 0.5 \cdot 0.8 = 0.2$.

This example indicates that the idea of combining Datalog with probabilities yields very powerful retrieval methods. However, if we want to apply probability theory consequently, then we soon run into difficulties. Assume that in our hypertext structure, we search for documents both about IR and DB (similar to `q1`):
```
q4(X) ← about(X,ir) , about(X,db).
```
Then simple multiplication of the probabilistic weights involved in the inference process would give us for document d2: $0.5 \cdot 0.7 \cdot 0.5 \cdot 0.8 = 0.14$. This is not correct, since the probability for the related-ness between d2 and d1 is considered twice; thus, the proper result would be 0.28. Besides counting the same probabilistic event twice, this simple approach also is unable to consider disjointness of complex events, for example when we search for documents either about IR or DB, but not about both:
```
q5(X) ← irnotdb(X).
q5(X) ← dbnotir(X).
irnotdb(X) ← indterm(X,ir) , ¬ indterm (X,db).
dbnotir(X) ← indterm(X,db) , ¬ indterm (X,ir).
```
If we would assume probabilistic independence of the subgoals of q5 (although they are disjoint events), we would compute the invalid result $0.7 \cdot 0.2 + 0.8 \cdot 0.3 - (0.7 \cdot 0.2) \cdot (0.8 \cdot 0.3) \approx 0.35$ instead of the correct probability $0.7 \cdot 0.2 + 0.3 \cdot 0.8 = 0.38$ for `q5(d1)`. The only way to overcome this problem in general is to switch from extensional semantics to intensional semantics (see e.g. [Pearl 88, pp. 4–12] for the comparison of these two approaches to uncertainty reasoning). For this purpose, we must keep track of the events that contribute to a derived fact.

In pD, we assume that each ground fact and each instantiated rule (i.e. where all variables are substituted by constants) corresponds to a basic (probabilistic) event, which can be either true or false. The pD program specifies the probability of the fact or rule being true. For any derived fact, in order to estimate the probability of being true, we have to consider the probabilities of ground facts and instantiated rules it depends on. In order to achieve this, we assign each ground fact and each instantiated rule a unique event key. Any derived fact relates to a Boolean combination of basic events from which this fact was inferred. Thus, we assign derived facts an event expression consisting of a Boolean combination of the event keys of ground facts and rules involved in its derivation. By using the event expression in combination with the probabilities of the basic events involved, we can compute the probability of the derived fact.

Throughout the examples given in this paper, we will use the first letter of the corresponding predicate (in case of rules possibly followed by an additional index) along with the argument constants as event keys. For derived facts, we will denote the event expression in brackets (omitting event keys of certain events). Thus, we have, for example,
```
q1(d1) [i(d1,ir) ∧ i(d1,db)]
q4(d2) [r(d2,d1) ∧ i(d1,ir) ∧ r(d2,d1) ∧ i(d1,db)]
q5(d1) [i(d1,ir) ∧ ¬ i(d1,db) ∨ ¬ i(d1,ir) ∧ i(d1,db)]
```
Given these Boolean expressions, we can identify identical events occurring more than once or disjoint events (e.g. the complement of an event). Then the corresponding probabilities can be computed correctly by means of the inclusion-exclusion-formula (see section 4).

3 Syntax and semantics

3.1 Syntax

We first explain the syntax of deterministic Datalog.

As basic elements, we have in Datalog *variables* (starting with capital letters), *constants* (numbers or alphanumeric strings starting with lower-case letters) and *predicates* (alphanumeric strings starting with lower-case letters).

Definition 1 *A* term *is either a variable or a constant symbol. If p is an n-ary predicate (with $n \geq 0$) and $t_1, \cdots, t_n$ are terms then $p(t_1, \cdots, t_n)$ is an* atom*. A* literal *is either an atom or a negated atom.* □

We use the word *ground* as a synonym for "variable-free". Since Datalog does not allow for functions in terms, a *ground term* can only be a constant, and the *Herbrand Universe* $\mathcal{U}$ of a Datalog program is the set of constants occurring in it. When we talk about "instantiated" atoms and rules, we mean that values from $\mathcal{U}$ are substituted for all variables in the atom or rule.

Definition 2 *A* rule *is a sentence of the form*

$$A \leftarrow L_1, \ldots, L_n$$

where A is an atom, and $L_1, \ldots, L_n$ are literals ($n \geq 0$). A is called the head *of the rule and $L_1, \ldots, L_n$ the* body *of the rule. Each L_i is a* subgoal *of the rule. All variables are assumed to be universally quantified at the front of the rule, and the commas in the body denote conjunction. If the body of a rule is empty, then we may refer to the rule as a* fact*, and omit the "$\leftarrow$" symbol.*

A Datalog program *is a finite set of rules.*

A query *is a conjunction of literals.* □

We call a Datalog program *(globally) stratified* if there is an assignment of ordinal levels to *predicates* such that whenever a predicate appears negatively in the body of a rule, the predicate in the head of that rule is of strictly higher level, and whenever a predicate appears positively in the body of a rule, the predicate in the head has at least that level. Stratified programs can be evaluated in a simple way by computing the extensions of predicates in the order of increasing ordinal levels (see e.g. [Ullman 88]). Furthermore, it can be checked syntactically (by computing the ordinal levels of predicates) whether or not a program is stratified. Unfortunately (as we will show later), there is a need for non-stratified programs in IR applications; these programs have a more complex semantics and require a more sophisticated evaluation strategy.

Probabilistic Datalog differs from deterministic Datalog only in that it allows for probabilistic weights to be attached to rules (and facts):

Definition 3 *A* pD rule *has the form αr, where r is a rule and α with $0 < \alpha \leq 1$ is the (probabilistic) weight of the rule. A weight of 1 can be omitted.*

A pD program *is a finite set of pD rules.* □

We distinguish between the deterministic part P_D and the indeterministic part P_I of a Datalog program. Then we can generate the set $D(P)$ of all determistic programs of a pD program P by forming all combinations of the determistic part and subsets of the indeterministic part.

Definition 4 *For a pD program P, let $H(P)$ denote the set of ground instances of all predicates from P (all possible ground atoms), and let P_i denote the set of all its instantiated rules.*

Furthermore, P_D is the set of all deterministic rules and P_I the set of all (instantiated) indeterministic rules from P_i (without the weights):

$$\begin{aligned} P_D &= \{r | 1r \in P_i\} \\ P_I &= \{r | \alpha r \in P_i \wedge \alpha < 1\} \end{aligned}$$

The set of all possible deterministic programs *of P is defined as (let $\mathcal{P}(X)$ denote the powerset of X):*

$$D(P) = \{P_D \cup Y | Y \in \mathcal{P}(P_I)\}$$

□

3.2 Semantics

For Datalog with negation, a number of different approaches have been described in the literature (see e.g. the survey in [Gelder et al. 91]). One of the most advanced approaches (that also yields the most plausible results) is the well-founded semantics described in [Gelder et al. 91]. Its basic idea is to use a three-valued semantics, where for each atom, a world may contain either the atom, its negation or none of both.

The definition of the well-founded model for a Datalog program Q is based on the notion of the *greatest unfounded set*. Informally speaking, given a partial interpretation of a program, this is the maximum set of ground literals that can be assumed to be false. The following definition starts with a set I of ground literals of predicates from the program Q; this set must be consistent, i.e. it does not contain both the positive and the negative literal for any ground atom. Then we compute the corresponding unfounded set as the set of all ground literals that we may conclude to be false. For this purpose, we consider the atoms occurring in the head of instantiated rules. Now there are two cases where we assume a head atom to be false:

i) the body of the rule obviously is false, due to one of its literals being false.
ii) the body contains another positive literal from the unfounded set, so we can assume both this body literal and the head atom to be false.

Definition 5 *Let H denote the set of ground instances H of predicates from Q and let I be a consistent set of ground literals whose atoms are in H. Then an* unfounded set *$A \subseteq H$ of Q (with respect to I) is a set of atoms such that for each $p \in A$ and each instantiated rule r of Q whose head is p,*

i) the complement of some literal in the body of r is in I or
ii) some positive literal in the body of r is in A.

The union of all unfounded sets w.r.t. I is itself unfounded and is known as the greatest unfounded set *$U_Q(I)$.* □

Based on the notion of an unfounded set, one can now define a transformation U_P that generates new negative atoms for a given set I, namely by negating all atoms in the unfounded set $U_Q(I)$; that is, we assume as many atoms as possible to be false. On the other hand, the transformation T_P generates new positive atoms only when we are forced to assume the truth, namely if the atom occurs in the head of a rule where the whole body is true.

Definition 6 *Transformations T_P, U_P, and W_P from sets of literals to sets of literals are defined as follows:*

- *$p \in T_P(I)$ if and only if there is some instantiated rule r of P such that r has head p and each literal in the body of r is in I.*
- *$U_P(I)$ is the greatest unfounded set of P with respect to I.*
- *$W_P(I) = T_P(I) \cup \neg \cdot U_P(I)$ (here $\neg \cdot U_P(I)$ denotes the negation of each atom in $U_P(I)$).* □

Since W_P is monotonic, it has a least fixpoint. This fixpoint is called the *well-founded model* of P.

The derivation of the well-founded model can be performed iteratively: Starting with $I = \{\}$, in each step negated atoms are added to I by means of the transformation $U_P(I)$, and new positive atoms are generated by means of $T_P(I)$.

Example 1 *As an example, consider the program consisting of the single rule*

```
p ← s, ¬ p.
```

Here the first iteration yields $I = \{\neg s\}$ (since there is no rule where s occurs in the head), and then, in the next iteration, we get the model $I = \{\neg s, \neg p\}$ due to condition i).

Example 2 *An example involving the second condition is the program*

```
p ← q, ¬ p.
q ← p, ¬ q.
```

yielding $I = \{\neg p, \neg q\}$ as model.

These two example have *complete models*, meaning that each instantiated atom of the program occurs either positively or negatively in the model (is either true or false). However, there are also programs that have only *partial models*:

Example 3 `p ← s, ¬ p.  s`

Here we get the model $I = \{s\}$, i.e. p is neither true nor false, since none of these values would satisfy the program.

In order to define a semantics for pD, we model a pD program as a probability distribution over the set of all possible deterministic programs. Then we can make use of the semantics defined for these programs.

Ideally, we could aim at using well-founded semantics for these programs. However, there are two major obstacles with this approach:

1. As discussed above, there are deterministic Datalog programs that have only a partial well-founded model. Now assume the following pD program:
   Example 4 `p ← s, ¬ p. 0.6 s.`
   This program can be modelled as a probability distribution over the two programs from example 1 and 3. Since the latter has a partial model only, the computation of the probability of p raises a problem: With probability 0.4, p is false, and with probability 0.6, it is undefined.
   So allowing for programs with partial well-founded models would force us to combine probability theory with a three-valued semantics. In order to avoid the complexity involved with this approach, we restrict to pD programs that can be modelled as probability distributions over deterministic programs with complete models.
2. Due to the definition of the greatest unfounded set, the computation of the well-founded model cannot always be performed by means of a subgoal-at-a-time evaluation. (E.g. in example 2, this strategy would jump between the evaluation of p and q and thus fall into an infinite loop.) Thus, the standard evaluation algorithms cannot be applied, and we would need a different, less efficient algorithm (e.g. the XSB system described in [Sagonas et al. 94] switches to interpreter mode as soon as it encounters a Datalog program which is not globally stratified). So we want to consider only programs that can be evaluated a-subgoal-at-a-time.

Taking these two criteria together, we want to consider only programs with a complete well-founded model that also can be evaluated a-subgoal-at-a-time. For deterministic Datalog, this class of programs has been defined as modularly stratified programs ([Ross 94]). Since the definition of modular stratification is rather complicated, the precise definitions are cited in the appendix only. Here we give a simpler explanation: In contrast to global stratification, modular stratification is formulated w.r.t. the instatiation of a program P for its Herbrand universe $\mathcal{U}$. The program P is modularly stratified if there is an assignment of ordinal levels to ground atoms such that whenever a ground atom appears negatively in the body of a rule, the ground atom in the head of that rule is of strictly higher level, and whenever a ground atom appears positively in the body of a rule, the ground atom in the head has at least that level.

Example 5 *The classical example for modular stratification ([Kolaitis 91]) is the game-playing program P consisting of the rule*
`win(X) ← move(X,Y), ¬ win(Y)`
together with some facts about `move`. `X` *is a winning position if there is a move from* `X` *to a position* `Y` *and* `Y` *is a losing position. For example, given the facts*
`move(4,3).  move(3,1).  move(3,2).  move(1,0).  move(2,0).`
winning positions are 1, 2 and 4.

Here `win(0)` *would be assigned ordinal level 1,* `win(1)` *and* `win(2)` *are on level 2,* `win(3)` *has level 3 and* `win(4)` *level 4. Obviously, this program is modularly stratified as long as the game graph is acyclic: Adding e.g. the fact* `move(3,4)` *would result in a cyclic graph, and thus an assignment of ordinal levels to the* `win` *facts as before would not be possible.*

As an IR example requiring modular stratification, consider the problem of retrieval of structured documents:

Example 6 *Let predicate* `part(D,P)` *state that* `P` *is a part of* `D`. *Each part may be assigned index terms via the predicate* `indterm(D,T)`. *A reasonable retrieval strategy would be to return a whole document (or a part) consisting of several parts only if all its parts are about the current topic. In Datalog, this can be formulated as:*

about(D,T) ← part(D,X) , about(X,T) , ¬ nabpart(D,T).
nabpart(D,T) ← part(D,P) , ¬ about(P,T).

The first rule states that D is about T if there is at least one part about T, and there is no part which is not about T. This Datalog program is not globally stratified, since about *refers recursively to itself involving negation. However, if the part structure is acyclic, then the program is modularly stratified and thus can be evaluated by our approach.*

Similar to this example, one can formulate retrieval rules when a hierarchic thesaurus is given, e.g. for assigning a broader term to a document in case all its narrower terms occur in it. So we see that acyclic structures are quite common in IR applications.

In general, checking if a program is modularly stratified is an NP-complete problem. However, as argued already in [Ross 94], modular stratification can be guaranteed by posing constraints on the instances of predicates, and then considering the rules (with non-empty heads) only. In our example, acyclicity of the part relation would be the corresponding constraint. In contrast, if we would consider hypertext links instead, then a cyclic link structure would violate the modular stratification.

For the semantics of pD, we use a possible worlds semantics by mapping a pD program onto a set of deterministic Datalog programs, where the model of each of the latter corresponds to a possible world. Then we can define the class of programs we want to consider:

Definition 7 *A pD program P is modularly stratified if every element of the set of its possible deterministic programs $D(P)$ is modularly stratified.* □

Obviously, any (deterministic) modularly stratified program also is a modularly stratified pD program. Thus, our approach is an extension over both modularly stratified Datalog and pD with global stratification.

Now we define the possible worlds semantics for a modularly stratified pD program.

Definition 8 *For a deterministic program Q with a complete, well-founded model, let $WF(Q)$ denote this model. For a (modularly stratified) pD program P, the set of possible worlds $\mathcal{W}(P)$ is defined as*

$$\mathcal{W}(P) = \{WF(P') | P' \in D(P)\}$$

□

Given the structure of the possible worlds for a pD program, we now turn to the interpretation of the probabilistic weights.

Definition 9 *For a (modularly stratified) pD program P with its set of possible worlds $\mathcal{W} = \mathcal{W}(P)$, let $M = (\mathcal{W}, \mu)$ denote a probability structure, where μ is a discrete probability distribution on $\mathcal{W}$. Let r denote a rule of the form $L_0 \leftarrow L_1, \ldots, L_n$. Furthermore, let v denote a valuation mapping variables occurring in P onto constants from the corresponding Herbrand universe $\mathcal{U}$. For a valuation v and a literal L, let $\underline{v}(L)$ denote the atom resulting from replacing the variables occurring in L by the corresponding constants, according to v. In a similar way, $\underline{v}(r)$ stands for the instantiated rule derived from r by applying v. Finally, let α denote a real number with $0 < \alpha \leq 1$.*

Then we can recursively define the notion of an extension $[t]_{(M,w,v)}$ *of a probabilistic term t for a valuation v and a world w of M (i.e. $\mathcal{W}$) and the notion of the* truth $(M, w, v) \models \phi$ *of a formula ϕ for a valuation v and a world w of M by means of the following rules:*

1. $[\alpha]_{(M,w,v)} = \alpha$
2. $[r]_{(M,w,v)} = \mu(\{w' \in \mathcal{W} : (M, w', v) \models r\})$
3. $(M, w, v) \models \alpha r$ *iff* $[\alpha]_{(M,w,v)} = [r]_{(M,w,v)}$
4. $(M, w, v) \models L$ *iff* $\underline{v}(L) \in w$
5. $(M, w, v) \models L_0 \leftarrow L_1, \ldots, L_n$ *iff whenever* $(M, w, v) \models L_1 \wedge \ldots \wedge (M, w, v) \models L_n$ *then* $(M, w, v) \models L_0$

□

Extensions are defined by the first two rules. Rule 1 states that the extension of a rational constant is always the rational number it represents. The second rule specifies that the extension of a rule is computed as the sum of the probabilities of the worlds in which the rule is true. In rule 3, the truth of probabilistic

rules of the form αr is defined by stating that the extension of the rule r must equal the extension of the probabilistic weight α. For example, for the probabilistic fact `0.4 indterm(d1,ir)`, rule 1 gives us the value 0.4, rule 2 computes the probabilities of the worlds containing the atom `indterm(d1,ir)`, and rule 3 requires that these two values are equal. The fourth rule states that a literal is true for a given valuation if the corresponding ground literal $\underline{v}(L)$ is an element of w. Finally, the truth of a rule is defined such that its body is true, then also the head must be true.

The last rule is specific for our approach, in that it refers to the specific valuation rather than to all possible valuations (due to the universal quantification). In order to point out the difference, let us consider an example:

Example 7 *The following program states that there is 50% chance that an arbitrary human is a woman, and that* `su` *and* `jo` *are human.*

`0.5 woman(X)` $\leftarrow$ `human(X). human(jo). human(su).`

If we would require that a rule is true only if it holds for all valuations, then a possible probability structure would be M_1*:*

$P(W_1) = 0.5$*:* {`woman(jo), woman(su), human(jo), human(su)`}
$P(W_2) = 0.5$*:* {¬`woman(jo), ¬woman(su), human(jo), human(su)`}

However, we would like to allow also for the following probability structure M_2*:*

$P(W_1) = 0.25$*:* {`woman(jo), woman(su), human(jo), human(su)`}
$P(W_2) = 0.25$*:* {`woman(jo),` ¬`woman(su), human(jo), human(su)`}
$P(W_3) = 0.25$*:* {¬`woman(jo), woman(su), human(jo), human(su)`}
$P(W_4) = 0.25$*:* {¬`woman(jo),` ¬`woman(su), human(jo), human(su)`}

As another possible interpretation of the probabilistic rule in the program, one could require that it always holds for the indicated proportion of all individuals, thus preferring the structure M_3*:*

$P(W_1) = 0.5$*:* {`woman(jo),` ¬`woman(su), human(jo), human(su)`}
$P(W_2) = 0.5$*:* {¬`woman(jo), woman(su), human(jo), human(su)`}

This example illustrates the difference between our approach and other possible interpretations of probabilistic rules. In principle, there are at least three interpretations:

1. The rule holds with the indicated probability for all possible valuations (as in M_1, but neither M_2 nor M_3).
2. The probability of the rule being true is computed for each specific valuation (thus allowing for e.g. M_1 and M_2).
3. The rule holds for the indicated proportion of all individuals in each possible world (as in M_3).

We prefer the second interpretation over the other two, since it gives more reasonable results and is more appropriate for IR applications. In the example from above, asking for the probability that both `jo` and `su` are women, only structure M_2 yields the preferred answer 0.25 — which, in turn, is only possible with the second interpretation.

Finally, we define the validity of a derived (probabilistic) fact based on the notion of a theory:

Definition 10 *A* theory of pD *is a set* Φ *of formulae which is closed under logical consequence; i.e.* Φ *is such that, if* $\phi \in \Phi$ *and* ϕ' *is true in all worlds in which* ϕ *is true, then also* $\phi' \in \Phi$*. A formula* ϕ *is* valid in a theory Φ *of pD, written* $\models_\Phi \phi$*, iff* ϕ *is true in all worlds* w *in which all formulae in* Φ *are also true.* □

4 Evaluation of pD programs

Our approach of evaluating pD programs is based on the notion of *event keys* and *event expressions*. As mentioned already in section 2, the probability of any derived fact has to be computed from the probabilities of the ground facts and instantiated rules that were used for deriving this fact. Thus, our basic *events* are instantiated rules (or ground atoms). Each derived fact also represents a probabilistic event. The dependency between this event and the underlying basic events can be described by means of a Boolean expression, i.e. the event of any derived fact can be expressed as a Boolean combination of basic events. For this purpose,

we assign each basic event an event key, and each derived fact is associated with an event expression that is a Boolean combination of the event keys of the underlying basic events.

Definition 11 *A set of* **event keys EK** *is a set of identifiers, that also contains the special elements* $\bot$ *(always false) and* $\top$ *(always true).*

For a pD program P*, let* $\varepsilon : P_i \rightarrow \mathbf{EK} - \{\bot\}$ *denote a mapping which assigns each instantiated rule* h *an event key* $\varepsilon(h)$*, with the following constraints:*

1. $\forall h \in P_i\ (h = 1g \wedge g \in H \Leftrightarrow \varepsilon(h) = \top)$
2. $\forall h, h' \in P_i\ (\varepsilon(h) = \varepsilon(h') \Rightarrow h = h' \vee h = 1g \wedge h' = 1g')$

Furthermore, for $\eta \notin \{\top, \bot\}$*, let* $\varepsilon^{-1}(\eta)$ *denote the inverse mapping* $\mathbf{EK} - \{\top, \bot\} \rightarrow P_i$. □

The first rule states that probabilistic heads (which also can be ground facts only) with weight 1 are assigned the event key $\top$, and the second rule requests that two different rules have the same event key only in case they have probability 1.

Definition 12 *For a pD program* P*, let* v *denote a valuation and* α *a probabilistic weight. Furthermore, let* F *denote a ground atom and* $L, L_0, L_1, \ldots$ *stand for literals. Then we can define a mapping from* P_i *onto the set of Boolean expressions over* **EK***, by means of the following transformations:*

1. $\eta(\underline{v}(L)) := \bigvee_s\ \eta(\underline{v}(s))$*, where* $s \in P$ *is a probabilistic rule whose head matches* $\underline{v}(L)$*.*
2. $\eta(\underline{v}(\alpha L_0 \leftarrow L_1, \ldots, L_n)) := \varepsilon(\underline{v}(\alpha L_0 \leftarrow L_1, \ldots, L_n)) \wedge \eta(\underline{v}(L_1)) \wedge \ldots \wedge \eta(\underline{v}(L_n))$*.*
3. $\eta(\underline{v}(L)) := \neg\eta(\underline{v}(|L|))$*, if* L *is a negative literal.*

Let **EE** *denote the set of all event expressions that can be formed this way.* □

Here the first transformation states that each rule whose head matches the current goal $\underline{v}(L)$ has to be applied, and the final event expression is formed as the disjunction of the event expressions resulting from these rules. The next transformation describes the forming of event expressions for pD rules, where we take the event key for the rule and the event expressions of the subgoals (if there are any). The last transformation forms the event expression of negative goals as the negation of the event expression of the corresponding positive goal.

Now we specify a mapping from event expressions onto sets of possible worlds which gives the set of worlds in which the fact corresponding to the event expression is true.

Definition 13 *For a pD program* P *with a set of possible worlds* $\mathcal{W}$ *and a set of event expressions* **EE***, let* $\mathcal{B}(\mathbf{EE}, \bot, \top, \{\wedge, \vee, \neg\})$ *denote the Boolean algebra over* **EE***. Then we can define a mapping* $\omega : \mathbf{EE} \rightarrow \mathcal{P}(\mathcal{W})$ *onto sets of possible worlds such that for any* $g \in H(P)$*,* $\omega(\eta(g)) = \{w | g \in w\}$*:*

1. $\omega(\top) = \mathcal{W}$*,*
2. $\omega(\neg\eta) = \mathcal{W} - \omega(\eta)$*,*
3. $\omega(\eta_1 \wedge \eta_2) = \omega(\eta_1) \cap \omega(\eta_2)$*,*
4. $\omega(\eta_1 \vee \eta_2) = \omega(\eta_1) \cup \omega(\eta_2)$*,*
5. $\omega(\eta) = \{w | \varepsilon^{-1}(\eta) = \alpha g \wedge g \in w\}$*, if* $\eta \in \mathbf{EK} - \{\top, \bot\}$. □

Thus we have shown that for any fact F derived by a pD program, the corresponding event expression $\eta(F)$ gives us via the mapping $\omega(\eta(F))$ the set of worlds in which F is true.

In principle, the probability of F being true can be computed as $\mu(\{w | w \in \omega(\eta(F))\})$ (see e.g. example 9 below). In our approach, we compute the probability based on the event expression, by exploiting the fact that we have a Boolean algebra on **EE**, and that we can apply the axioms from this algebra in order to transform event expressions.

Definition 14 *The probability of an event expression* $e \in \mathbf{EE}$ *is computed according to the formula*

$$P(e) = \mu(\omega(e)).$$

□

In the case of complex event expressions, the inclusion-exclusion formula ([Billingsley 79, p. 20]) can be used. This formula computes the probability of a disjunction of events in a set-theoretic way: For the union of two or more sets of events, first the sum of the probabilities of the sets is formed, from which the probability of the intersection has to be subtracted, e.g. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$. In case there are more than two sets, this strategy is applied recursively (for computing the probabilities of the intersections), e.g.

$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C)$. In order to apply this formula, we first have to transform the event expression into disjunctive normal form (DNF), that is:

$$e = K_1 \ \vee \ldots \vee \ K_n,$$

where the K_i are event atoms or conjunctions of event atoms, and an event atom is either an event key or a negated event key (n is the number of conjuncts of the DNF). From Boolean algebra, we know that any Boolean expression can be transformed into DNF.

Definition 15 *Let $e = K_1 \ \vee \ldots \vee \ K_n$ denote an event expression in disjunctive normal form. Then the probability for this expression is computed as follows:*

$$\begin{aligned} P(e) &= P(K_1 \ \vee \ldots \vee \ K_n) \\ &= \sum_{i=1}^{n} (-1)^{i-1} \left(\sum_{\substack{1 \le j_1 < \\ \ldots < j_i \le n}} P(K_{j_1} \ \wedge \ldots \wedge \ K_{j_i}) \right). \end{aligned} \tag{1}$$

□

For computing the probabilities for the AND-combination of conjuncts, we can simplify the corresponding expressions by means of the axioms of Boolean algebra. Combinations containing an event atom and its negation yield $\perp$ and thus can be ignored. Furthermore, duplicate event keys in a combination can be removed. This way, the probability of an event expression is finally computed as the sum of probabilities of conjunctions of event atoms.

Example 8 *The event expression for* `q5(d1)` *in section 2 leads to the following computation:*

$P($`i(d1,ir)` $\wedge \neg$ `i(d1,db)` $\vee \neg$ `i(d1,ir)` $\wedge$ `i(d1,db)`$) =$
$P($`i(d1,ir)` $\wedge \neg$ `i(d1,db)`$) + P(\neg$ `i(d1,ir)` $\wedge$ `i(d1,db)`$) -$
$P($`i(d1,ir)` $\wedge \neg$ `i(d1,db)` $\wedge \neg$ `i(d1,ir)` $\wedge$ `i(d1,db)`$) =$
$P($`i(d1,ir)` $\wedge \neg$ `i(d1,db)`$) + P(\neg$ `i(d1,ir)` $\wedge$ `i(d1,db)`$)$

As discussed before, there are now two possibilities for going on:

1. As in probabilistic logics, we can chose a careful interpretation of the probabilistic information available. Then we can compute a point value for the probability of an event expression only if the probabilities of all conjuncts formed in the inclusion-exclusion formula are given explicitly. Otherwise, only a probability interval can be given as a result (e.g. $[0.1, 0.5]$ in this example)
2. We use additional assumptions about the independence of events, thus allowing us to compute always a point value (e.g. 0.38).

Example 9 *As an example illustrating the two different approaches, consider the program P:*

```
0.6 indterm(d1,ir).  0.5 indterm(d1,db).
```

Theoretically, there is an infinite number of possible models that differ in the probability distributions over $D(P)$. Here we give three typical examples (possible worlds with zero probability are omitted):

M_1*:*
$P(W_1) = 0.3$*:* {`indterm(d1,ir),` $\neg$ `indterm(d1,db)`}
$P(W_2) = 0.3$*:* {`indterm(d1,ir), indterm(d1,db)`}
$P(W_3) = 0.2$*:* {$\neg$ `indterm(d1,ir), indterm(d1,db)`}
$P(W_4) = 0.2$*:* {$\neg$ `indterm(d1,ir),` $\neg$ `indterm(d1,db)`}

M_2*:*
$P(W_1) = 0.5$*:* {`indterm(d1,ir),` $\neg$ `indterm(d1,db)`}
$P(W_2) = 0.1$*:* {`indterm(d1,ir), indterm(d1,db)`}
$P(W_3) = 0.4$*:* {$\neg$ `indterm(d1,ir), indterm(d1,db)`}

M_3*:*
$P(W_1) = 0.1$*:* {`indterm(d1,ir),` $\neg$ `indterm(d1,db)`}
$P(W_2) = 0.5$*:* {`indterm(d1,ir), indterm(d1,db)`}
$P(W_3) = 0.4$*:* {$\neg$ `indterm(d1,ir),` $\neg$ `indterm(d1,db)`}

M_2 and M_3 are extreme cases, from which probabilistic logic would conclude

$$0.1 \leq P(\texttt{indterm}(\texttt{d1},\texttt{ir}) \wedge \texttt{indterm}(\texttt{d1},\texttt{db})) \leq 0.5.$$

If we assume independence of events, M_1 is the only possible model, which yields:

$$P(\texttt{indterm}(\texttt{d1},\texttt{ir}) \wedge \texttt{indterm}(\texttt{d1},\texttt{db})) = 0.3.$$

Probabilistic logic would only yield a point value in case we would specify the probability of the conjunct explicitly, e.g.

```
0.3 indterm(d1,ir) , indterm(d1,db).
```

For IR applications, the probabilistic logics approach — although theoretically attractive — does not seem to be appropriate: Since we have to deal with large amounts of data, only the probabilities of single events (but not for combinations) will be known in most cases. Due to the lack of additional probability information, this cautious approach would yield large probability intervals for most answers to a query (as in our example), from which it would be difficult to derive the top ranking answers – the only thing the user of an IR system is interested in.

In contrast, all classical IR models are based on default assumptions about the independence or disjointness of single events (i.e. terms — see the surveys in [Fuhr 92], [Wong & Yao 95] and [Crestani et al. 98]). Numerous evaluations have demonstrated the high retrieval quality resulting from these models.

For these reasons, we chose the approach based on default independence assumptions in order to develop an inference system for pD.

5 pD with independence and disjointness

5.1 Concept

We assume that by default, events are independent. Furthermore, it is possible to define sets of disjoint events.[1] Based on these assumptions, we describe probabilistic Datalog with independence and disjointness (pD_I) below.

Independence of events means that the probability of the conjunction of events equals the product of their probabilities. That is, for any set of independent events with keys $\eta_1, \ldots, \eta_n$,

$$P(\eta_1 \wedge \ldots \wedge \eta_n) = \Pi_{i=1}^{n} P(\eta_i).$$

This assumption is suitable for most IR applications. With respect to eqn (1), this means that we can compute the probability of a conjunct of event atoms as the product of the probabilities of the single event atoms. If the event atom is an event key, then we take the probability given with the corresponding probabilistic fact or rule, and in the case of a negated event key, the complement probability is to be taken.

Example 10 *For the event expression from example 9, we get*

$P(\texttt{i(d1,ir)} \wedge \neg \texttt{i(d1,db)}) + P(\neg \texttt{i(d1,ir)} \wedge \texttt{i(d1,db)}) =$
$P(\texttt{i(d1,ir)}) \cdot (1 - P(\texttt{i(d1,db)})) + (1 - P(\texttt{i(d1,ir)})) \cdot P(\texttt{i(d1,db)})$.

In addition to independent events, we also consider the case of disjoint events.

Example 11 *Assume that we have only uncertain information about the publication years of books, e.g.*

```
0.3 py(d1,82).  0.2 py(d3,89).
0.7 py(d1,83).  0.7 py(d3,90).
1.0 py(d2,85).  0.1 py(d3,91).
```

Here the publication year of d1 is either 82 or 83, the publication year of d2 is certainly 85, and for d3, it is either 89, 90 or 91. Obviously, the facts relating to one document represent disjoint events. In databases, this situation is called "imprecise attribute values".

[1] In principle, this approach allows for modelling of arbitrary dependencies of events, namely by giving the probabilities for all possible combinations of dependent events and declaring these combinations as being disjoint.

The model of this program is (negative atoms are omitted here):
$P(W_1) = 0.06$*:* {py(d1,82), py(d2,85), py(d3,89)}
$P(W_2) = 0.21$*:* {py(d1,82), py(d2,85), py(d3,90)}
$P(W_3) = 0.03$*:* {py(d1,82), py(d2,85), py(d3,91)}
$P(W_4) = 0.14$*:* {py(d1,83), py(d2,85), py(d3,89)}
$P(W_5) = 0.49$*:* {py(d1,83), py(d2,85), py(d3,90)}
$P(W_6) = 0.07$*:* {py(d1,83), py(d2,85), py(d3,91)}

Now a query for books published after 89
b89(D) ← py(D,Y) , Y > 89. ?- b89(D)
would yield the event expression [p(d3,90) ∨ p(d3,91)] *for* d3*. In terms of the inclusion-exclusion formula, we have*
P(p(d3,90) ∨ p(d3,91)) = P(p(d3,90)) + P(p(d3,91)) − P(p(d3,90) ∧ p(d3,91)).

When computing the probability for this expression, we must consider that the two events involved are disjoint, thus the probability of the conjunction is 0, and we get the correct result as the sum of the two event probabilities $0.7 + 0.1 = 0.8$ *(this equals the sum of the probabilities of the worlds* W_2, W_3, W_5, W_6 *where* b89(d3) *is true).*

In general, if two events with keys η_1, η_2 are disjoint, this means that there is no possible world in which the corresponding facts both are true (i.e. $\omega(\eta_1 \wedge \eta_2) = \emptyset$), and thus we have

$$P(\eta_1 \wedge \eta_2) = 0.$$

In the following, we assume that disjoint events may occur only with the same predicate, i.e. events relating to different predicates are always independent. In order to identify disjoint events, we have introduced in [Fuhr & Rölleke 97] the concept of a *disjointness key* for relations. The schema of a relation consists of a set of attributes (in pD, each predicate corresponds to a relation, where the attributes are identified by means of their position). For a probabilistic relation R with attribute set A, a disjointness key $K \subseteq A$ can be specified. Two tuples of R represent disjoint events if they have identical values for all attributes in K. In our example, the first argument of py would be the disjointness key; thus, tuples with the same document number represent disjoint events, whereas tuples with different document numbers correspond to independent events. With regard to our possible world semantics, the disjointness key is a relational key in each possible world.

In pD$_I$, we can declare disjointness keys for predicates in a similar way:

Definition 16 *A* pD$_I$ program *consists of a* declaration section *followed by a pD program. The declaration section contains for each* n*-ary predicate* p *occurring in the pD program a declaration of the form* #$p(d_1, \ldots, d_n)$*, where for* $i = 1, \ldots, n$*,* $d_i =$ dk*, if the* i*-th argument belongs to the disjointness key, and* $d_i =$ av *(non-key attribute value) otherwise. Declarations with* $d_1 = \ldots = d_n =$ dk *may be omitted.*

Given this declaration, two instantiated rule heads $p(a_1 \ldots, a_n)$ *and* $p(b_1 \ldots, b_n)$ *are disjoint iff*

$$\forall i\ 1 \leq i \leq n(d_i = \texttt{dk} \rightarrow a_i = b_i \wedge (\exists j\ 1 \leq j \leq n(d_j = \texttt{av} \wedge a_j \neq b_j))).$$

Two instantiated rules are independent iff their heads are not disjoint.

Here we have defined disjointness with respect to instantiated rule heads. Thus, it is possible to have two or more different instantiated rules with identical rule heads which are treated as being independent according to our definition. However, as we will show in section 6, this situation should be avoided, namely by formulating rule bodies which are disjoint in case we have identical heads.

Based on this declaration of disjointness, we can define the semantics of pD$_I$ programs:

Definition 17 *For a modularly stratified pD*$_I$ *program, the semantics according to definition 9 is further restricted by the following two constraints concerning the extension of terms:*

(6) For any pair h, h' *of instantiated rules with disjoint rule heads, the following holds:*
$\forall v\ \mu(\{w \in \mathcal{W} : (M, w, v) \models h \wedge h'\}) = 0$

(7) For any set $\{h_1, \ldots, h_n\}$ *of instantiated rules which are pairwise independent, the following holds:*
$\forall v\ \mu(\{w \in \mathcal{W} : (M, w, v) \models h_j \wedge \ldots \wedge h_n\}) = \Pi_{j=1}^{n} \mu(\{w \in \mathcal{W} : (M, w, v) \models h_j\})$.

With disjoint events, the computation of the probability of the conjunct of event atoms has to be modified slightly: After eliminating conjuncts containing an event key and its negation and also removing duplicate event atoms from the conjunct, disjoint event keys have to be considered. If a conjunct contains two disjoint event keys as unnegated event atoms, then it must be removed. In case there is more than one negated event key from a set of pairwise disjoint events (i.e. with identical values for the disjointness key) — possibly together with an unnegated event key from the same set, the corresponding probability is computed as follows: If there is an unnegated event key, then the probability is equal to the probability of this key. Otherwise, we have only negated event keys, and the probability is computed as the complement of the sum of the corresponding probabilities.

Example 12 *Assume that the event keys η_1, η_2, η_3 denote disjoint events. Then we can compute the following probabilities:*

$$\begin{aligned} P(\eta_1 \wedge \eta_2) &= 0 \qquad \textit{(remove conjunct)} \\ P(\bar{\eta}_1 \wedge \eta_3) &= P(\eta_3) \\ P(\bar{\eta}_1 \wedge \bar{\eta}_2) &= 1 - (P(\eta_1) + P(\eta_2)) \end{aligned}$$

5.2 Implementation

In our research group, we have implemented a pD$_I$ system called Hyspirit (*Hy*permedia *s*ystem with *p*robabilistic *i*nference for the *r*etrieval of *i*nforma*t*ion). The evaluation of pD$_I$ programs in Hyspirit is performed in two phases:

1. The first step is almost identical to that of the evaluation of deterministic Datalog. The only difference is the additional construction of the event expressions in parallel to the derivation of facts.
2. For the event expressions derived in the first phase, probabilities are computed as described before.

As basic evaluation strategy, we use the magic sets method for modularly stratified programs as described in [Ross 94]. For a given query, this method transforms a Datalog program into a set of relational algebra equations. Then the fixpoint of these equations is computed. In our case, we use a probabilistic relational algebra as described in [Fuhr & Rölleke 97]. In [Rölleke & Fuhr 97], we have shown that the fixpoint computed by the magic sets strategy for deterministic Datalog is identical to that of pD. Thus, ignoring the minor overhead for constructing the event expressions, the computational complexity of the first phase is identical to that of the evalauation of deterministic Datalog programs.

In some cases, the event expressions derived in the first phase may have an infinite length, as in the following example:

Example 13 `0.8 indterm(d1,ir). 0.9 indterm(d1,db).`
`0.5 link(d1,d2). 0.5 link(d2,d3). 0.5 link(d3,d1).`
`about(D,T) ← indterm(D,T).`
`about(D,T) ← link(D,D1) , about(D1,T).`
Here the query `?- about(D,ir)` *would produce an event expression of infinite length, due to the cyclic link structure:*
`i(d1,ir) ∨ l(d1,d2) ∧ l(d2,d3) ∧ l(d3,d1) ∧ i(d1,ir) ∨ ...`
However, due to the absorption law in Boolean algebra, this expression is equivalent to `i(d1,ir)`.

Generally speaking, when an event expression for a fact refers to the event expression for the fact itself, then the conjunct containing this event expression can be eliminated.[2] In [Rölleke & Fuhr 97], we describe this strategy in detail and show that it always yields correct results.

The evaluation time of the inclusion-exclusion formula grows exponentially with the number of conjuncts. Practical experimentation with Hyspirit has shown that the evaluation of about 10 or more conjuncts is not feasible. The only exception is with disjoint events, which lead to probabilities of zero for many of the expressions in the second sum of eqn (1); our evaluation algorithm for this formula is able to detect many

[2] Note that due to modular stratification, an atom cannot refer to its own negation

of these cases in advance and thus eliminates significant portions of the whole computation (e.g. if K_1 is a conjunct of disjoint events, then also all AND-combinations containing K_1 represent impossible events).

A major advantage of the magic sets strategy involved in the first evaluation phase is its efficient handling of external data e.g. stored in a relational database. Top-down evaluation like e.g. SLDNF resolution used for Prolog processes a tuple at a time and thus would generate an SQL query for every single tuple. In contrast, the magic sets strategy processes a set at a time and thus retrieves sets of tuples with a single call to the external database; however, only those tuples needed for answering the current Datalog query are retrieved from external storage.

6 Application

In addition to the examples presented in the previous sections, here we want to give some more examples which illustrate different features of our approach.

So far, we have considered only Boolean combinations of terms as IR queries. However, we can also express probabilistic query term weighting in pD$_I$. This is accomplished by regarding the terms in a query as disjoint events:

Example 14 `# qtw(dk).`
`0.4 qtw(db).  0.6 qtw(ir).`
`0.8 indterm(d1,db).  0.7 indterm(d1,ir).`
`q10(D)` $\leftarrow$ `qtw(X) , indterm(D,X).`
The query `?- q10(d1)` *yields the event expression*
`q(db)` $\wedge$ `i(d1,db)` $\vee$ `q(ir)` $\wedge$ `i(d1,ir)`.
Since `q(db)` *and* `q(ir)` *are disjoint, the result is computed like a scalar product of query and document, namely* $0.4 \cdot 0.8 + 0.6 \cdot 0.7$.
Thus, we can achieve the same weighting scheme as for example in the vector model or in INQUERY ([Turtle & Croft 91]). In [Rölleke & Blömer 97], we describe experiments using pD$_I$ for implementing different retrieval strategies and for considering additional information like document structure and hypertext links.

Now we discuss probabilistic rules and outline some problems in the formulation of pD programs with probabilistic rules

Example 15 *As a simple example for reasoning with chaining of probabilistic rules, assume that the probability of an arbitrary man liking sports (*`l-s`*) is 70% , but for women, it is only 40%. For* `jo`*, we only know that (s)he is human, and we ask for the probability that (s)he likes sports:*
`# sex(dk,av).`
`0.7 l-s(X)` $\leftarrow$ `sex(X,male).`
`0.4 l-s(X)` $\leftarrow$ `sex(X,female).`
`0.5 sex(X,male)` $\leftarrow$ `human(X).`
`0.5 sex(X,female)` $\leftarrow$ `human(X).`
`human(jo).`
This leads us to the following model (we omit negative atoms here):
$P(W_1) = 0.35$*:* {`sex(jo,male), l-s(jo)`}
$P(W_2) = 0.15$*:* {`sex(jo,male)`}
$P(W_3) = 0.20$*:* {`sex(jo,female), l-s(jo)`}
$P(W_4) = 0.30$*:* {`sex(jo,female)`}

For the query `?- l-s(jo)`*, Hyspirit derives the event expression* `l1(jo)` $\wedge$ `s(jo, male)` $\vee$ `l2(jo)` $\wedge$ `s(jo, female)`*, yielding* $0.7 \cdot 0.5 + 0.4 \cdot 0.5 = 0.55$ *(here we use* `l1` *and* `l2` *as event key for distinguishing between the two instantiated rules), which equals the sum of* W_1 *and* W_3.

If there are several probabilistic rules for a predicate which also have at least one subgoal in common, then these have to be formulated rather carefully, in order to achieve the desired result.

Example 16 *We want to state that two documents are semantically related when there is a link from one to another or when they are written by the same author:*

```
sameauthor(D1,D2) ← author(D1,X) , author(D2,X).
0.5 related(D1,D2) ← link(D1,D2).
0.2 related(D1,D2) ← sameauthor(D1,D2).
```

Now assume that we have two documents with a reference in between which were also written by the same author. What is the probability of relatedness in this case? Due to the disjunction of the two rules, the program from above would yield a probability of 0.6. However, this may not be the result that we want. If we view probabilistic rules as the specification of conditional probabilities, then the specification of $P(r|l)$ *and* $P(r|s)$ *says nothing about the probability of* $P(r|l \wedge s)$*. Thus, we should write rules in a way which allows us to specify all the probabilities involved, e.g.*

```
0.7 related(D1,D2) ← link(D1,D2) , sameauthor(D1,D2).
0.5 related(D1,D2) ← link(D1,D2) , ¬ sameauthor(D1,D2).
0.2 related(D1,D2) ← ¬ link(D1,D2) , sameauthor(D1,D2).
```

In general, if we want to formulate probabilistic rules for a predicate that depends on n different subgoals, then there may be up to 2^n rules for this predicate specifying all possible combinations. This situation corresponds to the link matrix in Bayesian inference networks (see next section), where the same number of probabilities has to be specified in this case.

This kind of rule formulation representing conditional probabilities requires negation in the rule body. If the rules are recursive, too, then we need modular stratification:

Example 17 *As a variant of example 6 about retrieval from structured documents (where* `part(D,P)` *states that* `P` *is a part of* `D`*), let us formulate the following rules*

```
0.8 about(D,T) ← indterm(D,T) , ¬ abpart(D,T).
1.0 about(D,T) ← indterm(D,T) , abpart(D,T).
1.0 abpart(D,T) ← part(D,P) , about(P,T).
```

Here a node `D` *is about a term* `T` *with a certain probability, if only the node, but none of its parts is about* `T`*, but with a higher probability, if also any of its parts are about* `T`*. Assuming an acyclic part structure, this program is modularly stratified.*

As an extension of pD, we introduce vague predicates. These predicates are a new type of builtin predicates (i.e. neither rule nor ground facts are given for these predicates). In contrast to other builtin predicates, however, vague predicates yield probabilistic events.

Example 18 *Assume that a PC shop offers the following models (tuples denote model name, CPU type, memory size, disk size and price):*

```
pc(m1,pI,16,1200,900).
pc(m2,pII,32,1200,1000).
pc(m3,pII,32,2400,1100).
```

Now a customer asks for a model with a price less than 1000:

```
?- pc(MOD, CPU, MEM, DISK, PRICE), PRICE < 1000
```

Obviously, it would not be appropriate to interprete this condition in a Boolean way. Rather, there is a certain probability that the customer finally will pay more than 1000 in case he gets a special bargain. This situation can be modelled by interpreting the condition `PRICE < 1000` *probabilistically by means of a vague predicate* $\hat{<}$*, that would yield the following results:*

```
1.00 ≂<(900,1000)
1.00 ≂<(950,1000)
0.99 ≂<(1000,1000)
0.90 ≂<(1050,1000)
0.60 ≂<(1100,1000)
```

Thus, formulating the query using this predicate

```
?- pc(MOD, CPU, MEM, DISK, PRICE), PRICE ≂< 1000
```

should yield the outcome

```
1.00 pc(m1,pI,16,1200,900)
0.99 pc(m2,pII,32,1200,1000)
0.60 pc(m3,pII,32,2400,1100)
```

Internally, the application of vague predicates generates new event keys that are considered in the final probability computation. Thus, for the last answer, the event expression would be `p(m3)` $\wedge$ $\hat{<}$*(1100,1000).*

Since vague predicates can hardly be defined as a set of probabilistic facts (due to the possibly infinite number of elements), we view them as a special form of builtin predicates; in deterministic datalog, builtin predicates (e.g. $=, <, \geq$) yield Boolean values, wheras the vague predicates of pD correspond to probabilistic events.

Vague predicates are essential for advanced IR applications. Besides vague fact conditions (as in this example), they can, for example, be used for proper name search or retrieval of OCRed text (based on string similarity). In [Fuhr 96], we argue that vague predicates are essential for the logical view on IR systems, in order to be able to answer queries irrespective of existing access structures (physical data independence). For example, searching for documents containing a certain phrase could be achieved by means of a vague predicate for phrase search (e.g. `?- phrase(information_retrieval,D)`), where the actual implementation of the phrase search is left to the underlying IR system (as Boolean combination of the single words, as adjacency search or based on syntactical analysis). Another important application area is multimedia IR, where the notion of similarity used in most of these systems can be interpreted as vague predicates; for example, in image retrieval, typical methods compute similarity values for color, contour and texture (see e.g. [Flickner et al. 95]).

As an alternative approach to image retrieval, we have combined Hyspirit with the IRIS image indexing system ([Hermes et al. 95]) which performs semantic indexing by inferring semantic concepts from syntactic features. IRIS has been applied successfully to the domain of landscape photos, where it detects basic concepts like e.g. water, sand, stone, forest, grass, sky and clouds. By subdividing an image into tiles, IRIS identifies the concepts occurring in a tile. Then, for each concept, adjacent tiles with the same concept are joined and finally the corresponding minimum bounding rectangle (MBR) is computed. In addition to the position and the size of the MBR, IRIS also computes the certainty with which the concept is identified. The output of the indexing process for an image is a list of objects, where each consists of a concept and the corresponding parameters. In HySpirit, each image object is represented as a fact of the form `imgobj(O,I,N,L,R,B,T)`, where `O` denotes the object id, `I` the id of the image, `N` the name of the concept (water, sand,...) and `L,R,B,T` are the coordinates of the MBR. The probabilistic weight of the fact gives the certainty with which the object was identified. Based on this representation, we can formulate queries for images with a certain content. As an example, the following query searches for images with water (lake, river, sea) in front of stone (rocks):

`?- imgobj(OA,I,water,L1,R1,B1,T1) , imgobj(OB,I,stone,L2,R2,B2,T2) & B1` $\leq$ `B2`

This type of query is only possible with Hyspirit due to its combination of predicate logic with probabilistic inference.

An important application of modular stratification is presented in [Fuhr & Rölleke 98], where we introduce probabilistic 4-valued Datalog (p4D). For execution, p4D programs are translated in probabilistic Datalog programs as presented in this paper. Due to the translation process, any recursive rule in p4D is mapped onto a rule involving negation of the head predicate, thus violating global stratification and requiring modular stratification.

In the same paper, we also describe experiments with a large text database, namely the AP subcollection of the TREC databases comprising all messages of the AP newswire from 1989. These 84,678 documents contain 259 MB of text. After indexing, storage in a relational database takes 284 MB (including the index). We were able to process the original first 150 TREC queries on this dataset, with an average response time of about 450 seconds on a 170 MHz SUN UltraSparc1.

In [Rölleke & Blömer 97], we describe the application of several retrieval strategies (formulated as logical

rules) for the CACM collection, including strategies for considering hypertext links (similar to the examples shown above). Like other researchers (not using logic-based IR methods), we were able improve retrieval effectiveness when using information about links between documents.

7 Comparison with other approaches

Our approach is a further development of the probabilistic relational algebra (PRA) presented in [Fuhr & Rölleke 97] (a similar probabilistic algebra has been described in [Lakshmanan et al. 97]). Since PRA is a generalization of ordinary relational algebra, all equivalences from this algebra also hold for PRA. Due to this fact, standard relational query languages like e.g. SQL could be used in combination with PRA. However, since PRA and these languages do not allow for recursion, recursive structures cannot be handled in this approach. This is a major drawback, since recursive structures (e.g. hierarchical thesauri, hypertext link structures, hierarchical document structures) will become very important in IR.

A solid theoretical foundation for probabilistic Datalog has been presented by Ng and Subrahmanian in [Ng & Subrahmanian 93] and [Ng & Subrahmanian 94]. Based on the idea of probabilistic logic, a system of linear inequalities is constructed for any derived fact during the inference process. Solving this system yields the corresponding probability interval. Intervals (instead of point probabilities only, as in our approach) also may be attached to facts and rules. Since it is also possible to state independence or disjointness of events, this approach is more expressive than pD. However, a major difference is the handling of rule disjunction.

Example 19 *Consider the following variation of our human-woman example from before:*

`0.5 woman(X)` ← `human(X). human(jo). woman(jo).`

Here our approach would yield `woman(jo)` *with certainty, due to the disjunction of the two* `woman`*-rules (where the latter has an empty body). In contrast, Ng and Subrahmanian require that whenever there is more than one rule leading to the same fact, the intersection of the probability intervals resulting from the different rules has to be formed. Thus, the empty interval would be the result in this case.*

Thus, their approach would be appropriate only if there is always only one rule for deriving a fact. Alternatively, one would have to use probability intervals (which is not appropriate for typical IR applications), and formulate rules in a very cautious way. Ng and Subrahmanian base their approach on stable semantics; unfortunately, even for deterministic Datalog, the computation of the stable model is already NP-complete — whereas modular stratification requires only polynomial time ([Gelder et al. 91]).

An alternative approach to probabilistic Datalog has been presented in [Lakshmanan & Sadri 94]. Here a more general approach to uncertainty handling has been taken, by distinguishing between belief and doubt, which do not have to sum up to 1, and for which intervals can be given. Furthermore, Lakshmanan and Sadri consider different cases of probabilistic independence: in addition to independence and disjointness, also positive and negative correlation as well as total ignorance can be assumed; of course, these additional modes lead to probability intervals. The interpretation of rules and their combination is the same as in our approach. Lakshmanan and Sadri show that for an important subclass of the programs under consideration, computation of the least fixpoint takes polynomial time only. Negation has not yet been considered within this framework.

In [Poole 93a] and [Poole 93b], a system for probabilistic Datalog is described along with an efficient evaluation algorithm. Here, events are independent by default, and disjointness of events must be stated explicitly. As major restrictions, negation is not supported, and in case there are several rules for the same predicate, the bodies must represent disjoint events. Thus, in terms of relational algebra, this approach neither supports difference nor projection.

Based on Bayesian inference networks as described in [Pearl 88] (and thus also on intensional semantics), the INQUERY system ([Turtle & Croft 91]) aims at a similar goal as pD_I. This system also allows for more powerful probabilistic inference than classic probabilistic IR models. However, INQUERY is still restricted to propositional logic. On the other hand, any inference network that can be formulated in INQUERY also can be expressed in pD_I (by means of probabilistic rules). In terms of efficiency, INQUERY clearly outperforms

Hyspirit — mainly due to its limited expressiveness. Furthermore, the actual implementation uses extensional semantics; thus, only tree-shaped inference structures are evaluated correctly.

A general probabilistic logic based on first order logics is described in [Halpern 90]. This approach allows not only for attaching probabilities (points or intervals) to arbitrary formulas, but also for nesting probability operators, e.g. stating that the probability that more than 70 % of all men like sports is 0.1. Furthermore, in the logic called $\mathcal{L}_3$, Halpern also allows for probabilistic statements for sets of individuals within a single world. For example, stating that 50 % of all humans are female would mean that in each single world, 50 % of all individuals known to be human are female (as in the probability structure M_3 of example 7). However, this approach is in conflict with the minimum model approach underlying all standard Datalog semantics: For the $\mathcal{L}_3$-type of semantics, we would have to assume in the model of example 15, that there are additional (unknown) individuals in each world, in order to have always 50 % women. So we think that this type of probabilistic semantics cannot be combined with Datalog.

Based on $\mathcal{L}_3$, a probabilistic terminological logic named P-MIRTL is presented in [Sebastiani 94]. Due to the restriction to terminological logics (instead of general first-order logics), the computational complexity of the inference process is reduced. The probabilistic part of P-MIRTL is more expressive than that of pD, but it also suffers from the general problem of probabilistic logics of giving only probability intervals for derived formulas. Furthermore, no evaluation method for P-MIRTL programs has been presented so far. In order to overcome these problems, in [Meghini et al. 98] a combination of a terminological logic for IR (called MIRLOG) with fuzzy logic is described; this approach has also been implemented.

There is also a large number of papers dealing with the application of rule-based approaches to IR, but without considering the intrinsic uncertainty and vagueness of IR. For example, the textbook [Parsaye et al. 89] describes a combination of IR, object-oriented databases, hypermedia and knowledge bases based on production rules. Datalog-like rules are used for forming queries in the hypertext model described in [Garg 88].

8 Conclusions and outlook

In this paper, we have presented a probabilistic version of Datalog and shown its suitability for IR. Since deterministic Datalog is a standard query language for deductive databases, pD can be used for querying integrated IR and database systems — supporting even vague fact queries and imprecise data. We have also shown that modular stratification significantly increases the expressiveness of probabilistic Datalog, especially in order to specify conditional probabilities for recursive rules.

In comparison to classical IR models based on propositional logic, only pD offers the expressiveness required for new IR applications. On the other hand, there is a number of more ambitious approaches for probabilistic inference than ours, but none of them seems to be applicable to IR problems, mainly due to problems of computational complexity. Thus, we think that pD is a good compromise between expressiveness of the representation language and efficiency of the inference process.

Our implementation of pD_I builds on the experience gained from deterministic Datalog systems. Thus, the first phase of the inference process is rather efficient, whereas the probability computation in the second phase suffers from the intrinsic complexity of probabilistic inference ([Cooper 90]). Our experiences so far show that for typical applications, this occurs rather rarely.

One may argue that in comparison to other IR systems, our approach is very inefficient. However, the expressiveness of these systems is rather limited. For example, for many typical IR queries (like some of the examples in section 2), the final probabilities can be computed correctly without considering the event expressions, thus avoiding any overhead in comparison to today's experimental systems (see [Fuhr & Rölleke 97] for a discussion of this point). Following these ideas, we are working on the development of query optimization methods for queries of this type.

A general problem of IR systems is that there may be a potentially large number of answers to a query, but mostly the user is interested only in the top-ranking elements. In contrast, database systems always

have to deliver the complete answer set. The latter assumption also underlies the query evaluation strategies for Datalog that are used in Hyspirit, i.e. all elements yielding a nonzero probability are considered. Thus, for many queries, e.g. a text search containing a frequent word or image retrieval based on some similarity operator, a significant portion of the database has to be processed. For reducing the number of elements to be considered for a query consisting of several conditions, one can either apply a horizontal or a vertical strategy: In the first case, it is assumed that elements are already ranked according to decreasing probabilities for single conditions; then the top-ranking elements for all conditions are processed in parallel until the requested number of best answers to the query is determined (see e.g. [Pfeifer & Fuhr 95], [Fagin 96]). The vertical strategy uses the most significant query conditions as filter and evaluates the remaining query conditions only for the filtered elements (see e.g. [Moffat & Zobel 96]). We are aiming at including these strategies in Hysprit.

Another area of further development is the expressiveness of pD, especially for deriving new probabilities. So far, we can only compute probabilities for Boolean combinations of single events. However, in many probabilistic IR models, conditional probabilities have to be derived from given probabilities (e.g. given $P(d)$ and $P(t, d)$, compute $P(t|d)$). Furthermore, one would like to derive probabilities from given deterministic facts, (e.g. given relevance feedback data, estimate the probability that a term occurs in a relevant document); a general strategy for solving this problem has been described in [Wüthrich 93].

Acknowledgements

I wish to thank Christian Altenschmidt, Achim Oberreuter and Claus-Peter Klas who implemented the probabilistic Datalog engine Hyspirit as part of their work for their diploma theses. My special thanks go to Thomas Rölleke for many fruitful discussions and the supervision of the students' works. The formulation of the semantics of pD_I is based on a suggestion by Umberto Straccia.

References

Billingsley, P. (1979). *Probability and Measure*. John Wiley & Sons, Inc, New York.

Ceri, S.; Gottlob, G.; Tanca, L. (1990). *Logic Programming and Databases*. Springer, Berlin et al.

Cooper, G. (1990). The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks. *Artificial Intelligence 42*, pages 393–405.

Crestani, F.; Lalmas, M.; van Rijsbergen, C.; Campbell, I. (1998). "Is this document relevant?...probably". A survey of probabilistic models in Information retrieval. *Computing Surveys 30*. (To appear).

Fagin, R. (1996). Combining Fuzzy Information from Multiple Systems. In: *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 216–226. ACM, New York.

Flickner, M.; Sawhney, H.; Niblack, W.; Ashley, J.; Huang, Q.; Dom, N.; Gorkani, M.; Hafner, J.; Lee, D.; Petkovic, D.; Steele, D.; Yanker, P. (1995). Query by Image and Video Content: The QBIC System. *Computer 28(9)*, pages 23–32.

Fuhr, N.; Rölleke, T. (1997). A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Transactions on Information Systems 14(1)*, pages 32–66.

Fuhr, N.; Rölleke, T. (1998). HySpirit — a Probabilistic Inference Engine for Hypermedia Retrieval in Large Databases. In: Schek, H.-J.; Saltor, F.; Ramos, I.; Alonso, G. (eds.): *Proceedings of the 6th International Conference on Extending Database Technology (EDBT), Valencia, Spain*, Lecture Notes in Computer Science, pages 24–38. Springer, Berlin et al.

Fuhr, N. (1992). Probabilistic Models in Information Retrieval. *The Computer Journal 35(3)*, pages 243–255.

Fuhr, N. (1996). Object-Oriented and Database Concepts for the Design of Networked Information Retrieval Systems. In: Barker, K.; Özsu, M. (eds.): *Proceedings of the Fifth International Conference on Information and Knowledge Management*, pages 164–172. ACM, New York.

Garg, P. (1988). Abstraction mechanisms in hypertext. *Communications of the ACM 31(7)*, pages 862–870.

van Gelder, A.; Ross, K.; Schlipf, J. (1991). The Well-Founded Semantics for General Logic Programs. *Journal of the ACM 38(3)*, pages 620–650.

Halpern, J. Y. (1990). An Analysis of First-Order Logics of Probability. *Artificial Intelligence 46*, pages 311–350.

Harman, D. (1995). Overview of the Second Text Retrieval Conference (TREC-2). *Information Processing and Management 31(03)*, pages 271–290.

Hermes, T.; Klauck, C.; Kreyš, J.; Zhang, J. (1995). Image Retrieval for Information Systems. In: *SPIE Proceedings Vol. 2420 (Storage and Retrieval for Image and Video Databases III)*. San Jose, CA, USA.

Kolaitis, P. (1991). The Expressive Power of Stratified Programs. *Information and Computation 90*, pages 50–66.

Lakshmanan, L.; Sadri, F. (1994). Probabilistic Deductive Databases. In: *Proc. Int. Logic Programming Symp., (ILPS'94)*. MIT Press, Ithaca, NY.

Lakshmanan, L.; Leone, N.; Ross, R.; Subrahmanian, V. (1997). ProbView: a flexible probabilistic database system. *ACM Transactions on Database Systems 22(3)*, pages 419–469.

Meghini, C.; Sebastiani, F.; Straccia, U. (1998). MIRLOG: A Logic for Multimedia Information Retrieval. In: Crestani, F.; Lalmas, M.; van Rijsbergen, C. (eds.): *Logic and Uncertainty in Information Retrieval: Advanced models for the representation and retrieval of information*. Kluwer Academic Publishers, Boston et al.

Moffat, A.; Zobel, J. (1996). Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems 14(4)*, pages 349–379.

Ng, R.; Subrahmanian, V. S. (1993). A Semantical Framework for Supporting Subjective and Conditional Probabilities in Deductive Databases. *Journal of Automated Reasoning 10*, pages 191–235.

Ng, R.; Subrahmanian, V. S. (1994). Stable Semantics for Probabilistic Deductive Databases. *Information and Computation 110*, pages 42–83.

Parsaye, K.; Chignell, M.; Khoshafian, S.; Wong, H. (1989). *Intelligent Databases*. Wiley, New York.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, San Mateo, California.

Pfeifer, U.; Fuhr, N. (1995). Efficient Processing of Vague Queries using a Data Stream Approach. In: *Proceedings of the 18th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 189–198. ACM, New York.

Poole, D. (1993a). Logic Programming, Abduction and Probability. *New Generation Computing 11(3)*, pages 377–400.

Poole, D. (1993b). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence 64*, pages 81–129.

van Rijsbergen, C. J. (1986). A Non-Classical Logic for Information Retrieval. *The Computer Journal 29(6)*, pages 481–485.

Rölleke, T.; Blömer, M. (1997). Probabilistic Logical Information Retrieval for Content, Hypertext, and Database Querying. In: Fuhr, N.; Dittrich, G.; Tochtermann, K. (eds.): *Hypertext — Information Retrieval — Multimedia (HIM). Theorien, Modelle und Implementierungen integrierter elektronischer Informationssysteme*, pages 147–160. Universitätsverlag Konstanz. http://ls1-www.cs.uni-dortmund.de/HIM97/.

Rölleke, T.; Fuhr, N. (1997). Probabilistic Reasoning for Large Scale Databases. In: Dittrich, K.; Geppert, A. (eds.): *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'97)*, pages 118–132. Springer, Berlin et al.

Ross, K. (1994). Modular Stratification and Magic Sets for Datalog Programs with Negation. *Journal of the ACM 41(6)*, pages 1216–1266.

Sagonas, K.; Swift, T.; Warren, D. (1994). XSB as an Efficient Deductive Database Engine. In: Snodgrass, R. T.; M., W. (eds.): *Proceedings of the 1994 ACM SIGMOD. International Conference on Management of Data.*, pages 442–453. ACM, New York.

Sebastiani, F. (1994). A Probabilistic Terminological Logic for Modelling Information Retrieval. In: Croft, W. B.; van Rijsbergen, C. J. (eds.): *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 122–131. Springer-Verlag, London, et al.

Turtle, H.; Croft, W. (1991). Evaluation of an Inference Network-Based Retrieval Model. *ACM Transactions on Information Systems 9(3)*, pages 187–222.

Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville (Md.).

Wong, S.; Yao, Y. (1995). On Modeling Information Retrieval with Probabilistic Inference. *ACM Transactions on Information Systems 13(1)*, pages 38–68.

Wüthrich, B. (1993). On the Learning of Rule Uncertainties and their Integration into Probabilistic Knowledge Bases. *Journal of Intelligent Information Systems 2*, pages 245–264.

A Definition of modular stratification

Here we cite the definition of modular stratification from [Ross 94].

Definition 18 *We say a predicate p* depends upon *a predicate q if there is a sequence of rules $r_0, \ldots, r_{n-1}$ with predicates $p_0, \ldots, p_{n-1}$ in the head, respectively, such that*

1. *$p = p_0$ and $q = p_n$, and*
2. *for $i = 1, \ldots, n$, p_i appears (positively or negatively) in the body of r_{i-1}.*

We say p depends on q through k negations if exactly k of the appearances of $p_1, p_2, \ldots, p_n$ in $r_0, \ldots, r_{n-1}$, respectively, are negative. We say p depends negatively *on q if p depends on q through at least one negation. A predicate p is* mutually recursive with *a predicate q if p depends upon q and q depends upon p.* □

Definition 19 *Let F be a component (i.e., a subset of the rules) of a logic program P. We say F is a* complete *component if for every predicate p appearing in the head of a rule in F,*

- *all rules in P with head p are in F, and*
- *if p is mutually recursive with a predicate q, then all rules in P with head q are in F.*

If the predicate p appears in the head of a rule in F then we say p belongs to *F. If the predicate q appears in the body of a rule in F, but does not belong to F, then we say q is* used by *F. If an atom A has predicate p, and p belongs to F, then we may say that A also belongs to F.* □

Definition 20 *(Reduction of a component) Let F be a program component, and let S be the set of predicates used by F. Let M be a two-valued interpretation over the universe $\mathcal{U}$ for the predicates in S.*

Form $I_{\mathcal{U}}(F)$, the instantiation of F with respect to $\mathcal{U}$, by substituting terms from $\mathcal{U}$ for all variables in the rules of F in every possible way. Delete from $I_{\mathcal{U}}(F)$ all rules having a subgoal Q whose predicate in S, but for which Q is false in M. From the remaining rules, delete all (both positive and negative) subgoals having predicates in S (these subgoals must be true in M) to leave a set of instantiated rules $R_M(F)$. We call $R_M(F)$ the reduction of F modulo M. □

Definition 21 *(Modular Stratification) Let $\prec$ be the dependency relation between components. We say the program P is* modularly stratified *if, for every component F of P,*

1. *There is a total well-founded model M for the union of all components $F' \prec F$, and*
2. *The reduction of F modulo M is locally stratified.* □