A System implementing Fuzzy Hypothetical Datalog[⋆]

Pascual Julián-Iranzo
Dept. of Information Technologies and Systems,
University of Castilla-La Mancha.
Email: Pascual.Julian@uclm.es

Fernando Sáenz-Pérez
Dept. of Software Engineering and Artificial Intelligence,
Complutense University of Madrid.
Email: fernan@sip.ucm.es

***Abstract*—This paper presents a system implementing a novel addition to a fuzzy deductive database: hypothetical queries. Such queries allow users to dynamically make assumptions on a given database instance, either by adding or removing data, without changing the instance. Further, since a fuzzy database includes fuzzy relations, these relations can also be changed with assumptions. This ability for dynamic change seamlessly enables writing "what-if" applications such as decision-support systems. Here, the new language Fuzzy Hypothetical Datalog is presented, along with an operational semantics and stratified inference. It has been implemented in a working system DES readily available on-line.**

I. INTRODUCTION

Making assumptions in foreseeing future possible scenarios is determinant for some applications such as trending, logistics, planning and scheduling. However, real-world data is often not as precise as required because in particular may refer to either unknown facts or approximations. For example, it is unknown whether a given product under development will satisfy customer's needs, and user geolocation with GPS and WFPS (WiFi Positioning Systems) provides approximate data for deducing user routes and habits, which can be used to provide them with specific services and products based on their behaviour. Developing applications with expressive queries on data for these kinds of scenarios can be more productive by using declarative techniques, such as rule-based systems, which raises the abstraction level and therefore eases programming. This is arguably one of the best advantages of logic programming for programming productivity, and it is gaining a renewed interest in both industry and academia (most likely, the most prominent system is SWI-Prolog [27] nowadays, backed by different companies for real world applications: pathwayslms.com/swipltuts; see also the Strange Loop 2019 conference thestrangeloop.com). Thus, this work proposes a logic-based approach for embodying assumptions in a fuzzy deductive database.

Datalog [1] is the *de-facto* query language for deductive databases, which has regained interest in last years for commercial applications since efficient solving methods have been developed such as SAT (propositional SATisfiability problem) solvers (e.g., [10]). Several commercial systems with Datalog-like languages are available, such as LogicBlox [4], [11] (logicblox.com), Semmle [5] (semmle.com), Datomic (datomic.com) and DLV [24] (dlvsystem.com). As well, free, open-source systems exist such as pyDatalog for Python (sites.google.com/site/pydatalog), Datalog for Racket (docs.racket-lang.org/datalog), and Datalog Educational System (DES) [20] (des.sourceforge.net). Datalog was conceived as a query language for relational databases, more expressive than relational algebra and calculi because recursive definitions are allowed. With respect to SQL, it adds neater formulations and allows for non-linear and mutual recursion, features which are important to solve graph problems as those occurring in social networks [9].

Fuzzy Datalog (as can be found as early as [2]) embraces proposals coming from both deductive databases and fuzzy logic programming [12], [13], [17], [19], [23], [26]. Among other proposals, such as [18] (which explicitly include the computation of the rule approximation degree as a burden to the programmer), FuzzyDES (*FDES* for short) [15] extends the Datalog language in DES with fuzzy relations and predicates. In this system (which follows Bousi∼Prolog (*BPL*) [13], [19]), a Weak SLD (WSLD) resolution algorithm using fuzzy relations (described as sets of either proximity or similarity equations) is applied to compute the approximation degree to a query, replacing the classical Prolog SLD resolution (Selected linear resolution with Definite clauses) [16]. Additionally, each rule can be tagged with a weight that modulates its confidence, being known as a graded rule. This allows for solving database applications such as recommender systems [15] based on subjective data. The rest of this paper focuses on the system FuzzyDES.

Hypothetical Datalog was also proposed time ago [6], allowing inferences in hypothetical contexts where new database tuples (hypotheses) can be added and removed. Further works [21], [22] extended this setting by allowing for the assumption of both new rules and predicates, the notion of restricted predicates for pruning predicate semantics, strong integrity constraints, and duplicates for dealing with multiple copies of data as needed in aggregations. These additions were motivated by applications such as OLAP (Online Analytical

[⋆]This work has been partially supported by the EU (FEDER) and the State Research Agency (AEI) of the Spanish Ministry of Economy and Competition under grants TIN2016-76843-C4-2-R (AEI/FEDER, UE – project MERINET), TIN2013-44742-C4-3-R (project CAVI-ART) and Madrid Regional Administration under grants P2018/TCS-4339 (BLOQUES-CM) and S2013/ICE-2731 (N-GREENS Software-CM).

Processing), data warehouses, business intelligence and e-commerce [21]. As well, we focus on Hypothetical Datalog as implemented in DES.

In this work, results from both Fuzzy Datalog and Hypothetical Datalog are integrated into a deductive database with the aim of facing applications in which data is not crisp, knowledge can be modulated by approximations degrees, and "what-if" queries are relevant for making deductions in hypothetical scenarios. To this end, hypothetical inference is exported to a fuzzy deductive database, allowing for assumptions of facts, rules, and proximity equations. Since negative assumptions are also of interest in applications, the concept of restricted predicates [22] is adjusted to the fuzzy setting. Similarly, assumptions on proximity equations are proposed, as well as their hypothetical tuning. To the best of our knowledge, this is the first proposal of such an integration of features. As a practical outcome of this proposal, the system FuzzyDES implements this integration, and it has been made available as a working system with both desktop applications for multiple operating systems (des.sourceforge.net), and an on-line system for rapid testing (desweb.fdi.ucm.es).

A whole section is devoted to motivate by examples what is expected from this work (Section II). The next sections recall concepts and examples in both fuzzy deductive databases (Section III) and Hypothetical Datalog (Section IV). Then, their integration is proposed in Section V, followed by a description of its implementation in Section VI (which resorts to tabling, a technique based on dynamic programming). Finally, Section VII concludes and provides points for future work.

II. ASSUMPTIONS IN A FUZZY DATABASE

As a motivation for the integration of assumptions in a fuzzy deductive database, let us consider the following example in the stock market, following Prolog syntax:

```
stock_up(google) with 0.9.
stock_up(greek_bonds) with 0.2.

shareholder(paul,google).
shareholder(paul,greek_bonds).

keep_stock(Name,Stock) :-
  shareholder(Name,Stock), stock_up(Stock).
```

where `google` stock is expected to raise with a degree of 0.9 and `greek_bonds` with 0.2, and `paul` has shares of both `google` and `greek_bonds`. Then, the query `keep_stock(Name,Stock)` would return the stocks that are profitable to keep for each shareholder:

```
FDES> keep_stock(N,S)
Info: Processing: answer(N,S) :- Query.
{ answer(paul,google) with 0.9,
  answer(paul,greek_bonds) with 0.2 }
```

Observe that the system prompt is `FDES`, which is an indication of the fuzzy system mode (enabled with the command `/system_mode fuzzy`). Also, as in other database systems, the system automatically adds the default relation name `answer` for displaying the results. From here on and for the sake of space, we replace the body of `answer` by `Query`, where `Query` is the query posed at the top-level. Here, the second line would be read as `answer(N,S) :- keep_stock(N,S).`

A. Assuming Graded Rules

Now, let us assume that Paul is willing to buy Amazon shares with a stock up expectation of 0.7. Determining what stocks are profitable under this assumption would be expressed with the hypothetical goal:

```
FDES> stock_up(amazon) with 0.7 /\
      shareholder(paul,amazon) => keep_stock(N,S)
Info: Processing: answer(N,S) :- Query.
{ answer(paul,google) with 0.9,
  answer(paul,amazon) with 0.7,
  answer(paul,greek_bonds) with 0.2 }
```

It would be interesting to tune existing information in the database: On the one hand, by *raising* approximation degrees in graded rules. For example, if confidence on Greek bonds is expected to raise from 0.2 to 0.5:

```
FDES> stock_up(greek_bonds) with 0.5 =>
        keep_stock(N,S)
Info: Processing: answer(N,S) :- Query.
{ answer(paul,google) with 0.9,
  answer(paul,greek_bonds) with 0.5 }
```

On the other hand, tuning can be done by *lowering* such approximation degrees. For example, for testing a decreasing confidence in Google, it would not be appropriate to simply assume something as `stock_up(google) with 0.6`, because there is already a fact (`stock_up(google) with 0.9`) with a higher grade in the database, a fact that is not overwritten by this assumption:

```
FDES> stock_up(google) with 0.6 => stock_up(google)
Info: Processing: answer :- Query.
{ answer with 0.9 }
```

The resulting approximation degree is computed as the maximum of the approximation degrees of the two involved rules [15]. Thus, a decreasing tuning introduces a new requirement neither present in the fuzzy deductive database nor in the Hypothetical Datalog system, but in their integration. Restricting rules are used to this end as follows:

```
FDES> -stock_up(google) with 0.3 => keep_stock(N,S)
Info: Processing: answer(N,S) :- Query.
{ answer(paul,google) with 0.6,
  answer(paul,greek_bonds) with 0.2 }
```

Assuming a restricted graded rule (in the example, a fact) on a database (context) equates to decrease the maximum degree of the positive rule with the degree of the restricted rule in the current context for building the set of axioms. If more than one restricting rule exist for the same positive rule, the maximum grade is chosen to apply the reduction. This will be formalized in Section V.

B. Assuming Fuzzy Equations

Fuzzy relations are also subject to assumptions that change their definition by either adding or replacing existing fuzzy equations. For example, let us consider a fuzzy relation `near` that states how "near" are two cities by train (railway paths are generally not straight, and two cities can be linked by a much more train distance than it might be expected because of the orography):

```
:- fuzzy_relation(near,
         [reflexive,symmetric,transitive(product)]).
madrid      near ciudad_real = 0.6.
ciudad_real near badajoz     = 0.4.
badajoz     near lisbon      = 0.5.
```

This fuzzy relation has attached the reflexive, symmetric and transitive properties, with the product t-norm. Indeed, the train connection between `madrid` and `badajoz` is deceptive, but a new high-speed railway is under discussion for its development. If built, reaching `lisbon` would be faster:

```
FDES> madrid near lisbon
Info: Processing: answer :- Query.
{ answer with 0.12 }

FDES> madrid near badajoz = 0.4 => madrid near lisbon
Info: Processing: answer :- Query.
{ answer with 0.2 }
```

The new (explicit) `equation madrid near badajoz = 0.4` has been considered as an assumption. Before it was, the link between `madrid` and `badajoz` was still available because `near` is transitive, and its t-closure included `madrid near badajoz = 0.24` (it can be tested with `/list_t_closure near`).

By (automatically) recomputing the t-closure of `near` in the presence of the new explicit equation, it includes the implicit equation `madrid near lisbon = 0.2`, which is the one used to compute the answer.

In addition, it is interesting to replace an existing explicit equation with an assumption. Only one proximity equation between two given nodes is allowed in a database (adding a new one for the same nodes overwrites the older one). So, assuming an existing equation with a different approximation degree amounts to removing it in the assumption context and adding the new equation. Out of this context, the original equation remains. In the same example, considering that a new high-speed railway has been built between `ciudad_real` and `badajoz` (so that both cities are "closer"):

```
FDES> ciudad_real near badajoz = 0.6 =>
      madrid near lisbon
Info: Processing: answer :- Query.
{ answer with 0.18 }
```

Note that the assumed equation is only applicable in its hypothetical context, and it has no sense outside:

```
FDES> (ciudad_real near badajoz = 0.6
       => madrid near lisbon),
      ciudad_real near badajoz = Degree
Info: Processing: answer(Degree) :- Query.
{ answer(0.4) with 0.18 }
```

The second goal, which allows for accessing the degree corresponding to the enquired proximity equation, makes clear that, out of the hypothetical context, this degree is not altered (`Degree` is bound to 0.4).

Finally, note that removing an existing equation is equivalent to asserting (or assuming) another one between the same two elements with approximation degree 0.

III. FUZZY DEDUCTIVE DATABASES

Following [15], a fuzzy deductive database is composed of a set of (binary) fuzzy relations, a set of graded facts (the extensional database) and a set of graded rules (the intensional database).

A binary fuzzy relation $\mathcal{R}$ on a universal domain of discourse U defines the mapping $U \times U \longrightarrow [0,1]$, which may have attached several properties, namely: reflexive ($\mathcal{R}(x,x) = 1$), symmetric ($\mathcal{R}(x,y) = \mathcal{R}(y,x)$), and transitive ($\mathcal{R}(x,z) \geq \mathcal{R}(x,y) \triangle \mathcal{R}(y,z)$, where $\triangle$ is an arbitrary t-norm). $\mathcal{R}$ is a proximity relation if it is both reflexive and symmetric (e.g., useful for expressing closeness) $\mathcal{R}$ is said to be a similarity relation if, in addition, it is transitive (e.g., useful for expressing likeness). Symbols in U include both data terms (symbolic constants) and predicate symbols as disjoint signatures which cannot be related between them by $\mathcal{R}$. A fuzzy relation is specified both extensionally, with fuzzy equations of the form $\mathcal{R}(x,y) = \alpha$ (where α is the approximation degree between x and y), and intensionally, with the aforementioned properties. Each fuzzy program has attached a default similarity relation $\sim$ as an infix operator, which is used along WSLD to unify data, on the one hand, and predicate symbols, on the other hand. Thus, this default fuzzy relation is extended to atomic formulas as: $\mathcal{R}(p(t_1,\ldots,t_n), q(s_1,\ldots,s_n)) = \mathcal{R}(p,q) \triangle \mathcal{R}(t_1,s_1) \triangle \cdots \triangle \mathcal{R}(t_n,s_n)$, where p and q are $n-$ary predicate symbols, and t_i, s_i are either constants or variables. Following Prolog syntax, variables start with uppercase, while constants and predicate symbols start with lowercase. For example, given the proximity equations $tall/1 \sim medium/1 = 0.4$ and $ann \sim anne = 0.7$, and the t-norm minimum (i.e., Gödel), then $tall(ann) \sim medium(anne) = 0.4$.

A weak unifier between expressions $\mathcal{E}_1$ and $\mathcal{E}_2$ is defined by $\mathcal{E}_1\theta \sim \mathcal{E}_2\theta$, which must be greater than or equal to a given approximation degree threshold $\lambda \in [0,1]$ (known as λ-cut) specified for each program. The notion of weak most general unifier θ between two expressions, denoted by $\mathsf{wmgu}^{\lambda}_{\mathcal{R}}(\mathcal{E}_1, \mathcal{E}_2)$, is accordingly defined as the substitution such that there is no a more general weak unifier between $\mathcal{E}_1$ and $\mathcal{E}_2$. As well, a weak unification algorithm is provided with the definition of a transition system on $\sim$ (see [15] for details). For the same example equations above, $tall(ann) \sim medium(X) = 0.4$, with $\theta = \{X/ann\}$ (i.e., X is bound to ann).

A fuzzy program is a fuzzy theory defined with the mapping $\mathcal{F} : \Pi \longrightarrow (0,1]$,[1] where Π is the set of possible logic rules. In practice, a program can be seen as a set of graded logic rules. A graded logic rule takes the form $\langle A \leftarrow Q; \alpha \rangle$, where A (known as head) is an atomic formula or atom (i.e., an n-ary predicate, $n \geq 0$, applied to n terms),[2] $Q \equiv Q_1 \wedge \ldots \wedge Q_n$ is a conjunctive body composed of Q_i atoms, and α is the rule approximation degree (grade) applied by $\mathcal{F}$ to the logic rule. A fact is a graded logic rule with a true consequent, and it

[1]Note that 0 does not belong to this interval as it would mean that a rule has no support: it would be equivalent to omit the rule from the program.

[2]The case $n = 0$ corresponds to a propositional formula.

is simplified as $\langle A;\alpha\rangle$. A safe database requires ground facts and safe rules [15], [25]. An operational semantics for WSLD resolution has been defined for finding answers to queries [15]. In particular, the following inference rule represents a transition step $\Rightarrow_{\textit{WSLD}}$ for a conjunctive goal, where an atomic conjunctor A' is replaced by the body of a rule whose head weakly unifies with A':

$$\langle(\leftarrow A' \wedge Q'), \theta, \alpha\rangle \Rightarrow_{\textit{WSLD}} \langle\leftarrow (Q \wedge Q')\sigma, \theta\sigma, \delta\triangle\beta\triangle\alpha\rangle$$

if there exits a rule $\langle(A \leftarrow Q);\delta\rangle$ in the program, $\delta \geq \lambda$, $\sigma = \mathsf{wmgu}^{\lambda}_{\mathcal{R}}(A,A') \neq \mathit{fail}$, the unification degree $\beta = \mathcal{R}(A\sigma,A'\sigma) \geq \lambda$, and $(\delta\triangle\beta\triangle\alpha) \geq \lambda$ (a transition step can be applied if the computed approximation degree surpasses the threshold λ).

Example 1: Let us consider the proximity equations $\mathit{rent}/2 \sim \mathit{lease}/2 = 0.6$ between predicates (the arity N is added as $/N$ after each predicate symbol name), $\mathit{so\_much} \sim \mathit{very\_much} = 0.5$ between constants, the program Π

$$\{\langle(\mathit{likes}(\mathit{john}, \mathit{porsche}, \mathit{so\_much})); 1.0\rangle,$$
$$\langle(\mathit{rent}(P, C) \leftarrow \mathit{likes}(P, C, \mathit{very\_much})); 1.0\rangle,$$
$$\langle(\mathit{borrow}(P, C) \leftarrow \mathit{lease}(P, C)); 0.7\rangle\},$$

and the query $(\leftarrow \ \mathit{borrow}(P, C))$. Then, the following sequence of transition steps can be found: $\langle(\leftarrow \ \mathit{borrow}(P, C)), \emptyset, 1.0\rangle \Rightarrow_{\textit{WSLD}} \langle(\leftarrow \ \mathit{lease}(P, C)), \emptyset, 0.7\rangle \Rightarrow_{\textit{WSLD}} \langle(\leftarrow \ \mathit{likes}(P, C)), \emptyset, 0.6\rangle \Rightarrow_{\textit{WSLD}} \langle\square, \{P/\mathit{john}, C/\mathit{porsche}\}, 0.5\rangle$. Observe that $\mathit{lease}(P, C) \sim \mathit{rent}(P, C) = 0.7$, $\mathit{rent}(P, C) \sim \mathit{lease}(P, C) = 0.6$, and $\mathit{so\_much} \sim \mathit{very\_much} = 0.5$ and successive t-norm compositions become $1.0\triangle0.7\triangle0.6\triangle0.5 = 0.5$. The system proceeds as follows:

```
FDES> borrow(Person,Car).
Info: Processing:
  answer(Person,Car) :- borrow(Person,Car).
{ answer(john,porsche,0.5) }
```

$\square$

IV. HYPOTHETICAL DATALOG

Hypothetical Datalog allows for making two types of assumptions: positive knowledge for adding data, and negative knowledge for removing data when trying to prove a goal. The notion of embedded implication [7] is introduced to this end in order to represent hypothetical implications in rule bodies, as in $A \leftarrow (C \Rightarrow B)$: "Intuitively, in the formula $A \leftarrow B$, the atom B is 'executed' in order to prove A, while in the formula $C \Rightarrow B$, the atom C is 'assumed' to be true in order to prove B." So, C is assumed to be true in the context of any derivation starting at the goal B. Following [21], which builds on [7], the syntax of the language is extended with this kind of hypothetical goals, in particular with:

$$\phi ::= A \mid R_1 \wedge \ldots \wedge R_m \Rightarrow \phi$$

where A is an atom, ϕ is a goal, and each R_i is a rule.

An inference system for Hypothetical Datalog (defined by a relation $\Rightarrow_{\textit{DL}}$) introduces the notion of contexts in derivations, and the following rule is included in the inference system as part of its operational semantics:

$$\frac{\Delta \cup \{R_1, \ldots, R_n\} \vdash \phi}{\Delta \vdash R_1 \wedge \ldots \wedge R_n \Rightarrow \phi}$$

where ϕ is a goal, and Δ is a database. This inference rule means that if the inference expression above the line can be inferred, then the one below the line can be inferred too.

Intuitively, proving the embedded implication is equivalent to proving the consequent where the database has been extended which each one of the rules in the antecedent. An extended database is known as a context, which is only used in the inferencing process of the goal ϕ. This process considers, in particular, the following inference rule, which takes a matching rule to infer a goal: For any rule $A \leftarrow \phi_1 \wedge \ldots \wedge \phi_n$ in Δ, where $\mathit{str}(\Delta, \mathit{pred}(A))$ is the stratum (as explained next) corresponding to the predicate for A, and for any ground substitution θ:

$$\frac{\Delta \vdash \phi_i\theta \text{ for each } i}{\Delta \vdash A\theta}$$

where we omit duplicates with respect to [21] and therefore elide both rule and axiom source identifiers.

Stratification [25] is a well-known syntactical constraint to classify valid programs in terms of predicate dependencies, where each predicate is mapped to a stratum. In turn, it is based on the notion of predicate dependency graph, which includes a positive arc from each goal predicate in a body to the head predicate. If the goal is negated, the arc is negative. A stratum contains predicates which are not related by a negative arc in the transitive closure of the dependency relation. Stratification is adopted in [22] to enable negative knowledge in the form of restricting axioms (which is an axiom with a restricting atom denoted as $-A$), and dependencies are defined as:[3] A predicate P *positively* (*negatively*, respectively) depends on Q if P is the predicate symbol of A in a rule (both a program rule and a rule in a premise) $A \leftarrow G_1 \wedge \ldots \wedge G_n$, and Q occurs either in some positive (restricting, respectively) atom G_i, or in G in an embedded implication $G_j \equiv R_1 \wedge \ldots \wedge R_n \Rightarrow G$. A query Q to a database Δ can be specified as $\mathit{answer}(\overline{X}) \leftarrow Q$, where $\overline{X}$ are the relevant variables for the query.

Example 2: Let us consider the database Δ_0 composed of the Hypothetical Datalog rules $\{\mathit{own}(\mathit{john}, \mathit{umbrella}), \mathit{own}(\mathit{ann}, \mathit{umbrella}), (\mathit{walk}(\mathit{Person}) \ \leftarrow \ \mathit{raining}, \mathit{own}(\mathit{Person}, \mathit{umbrella}))\}$, and the query $Q \equiv -\mathit{own}(\mathit{john}, \mathit{umbrella}) \wedge \mathit{raining} => \mathit{walk}(\mathit{Person})$.[4] Since $\mathit{own}/2$ becomes a restricted predicate in $\Delta_0 \cup \{(\mathit{answer}(\mathit{Person}) \ \leftarrow \ Q)\}$, a possible stratification is $\{\mathit{own}/2 \ : \ 1, \mathit{raining}/0 \ : \ 1, \mathit{walk}/1 \ : \ 2, \mathit{answer}/1 \ : \ 2\}$, where predicates are characterized by their arity, and the mapping from a predicate to its stratum is denoted by the colon. Each stratum has associated an inference system, and the unified stratified semantics proceeds by strata, from lower to upper [21], in order to infer the meaning of databases and goals. Thus, for stratum 1, the following expressions can be inferred: $\{\Delta_0 \vdash \mathit{own}(\mathit{john}, \mathit{umbrella}), \Delta_0 \vdash \mathit{own}(\mathit{ann}, \mathit{umbrella}), \Delta_1 \vdash -\mathit{own}(\mathit{john}, \mathit{umbrella}), \Delta_1 \ \vdash \ \mathit{raining}\}$. And for stratum 2: $\{\Delta_1 \vdash \mathit{walk}(\mathit{ann}), \Delta_1 \vdash \mathit{answer}(\mathit{ann})\}$. Adapting the definition of the meaning of a goal [21], [22]: $\mathit{solve}(\phi, \mathcal{A}) = \{\Delta \vdash \psi \in \mathcal{A}$ such that $\phi\theta = \psi\}$, where ϕ is a goal, $\mathcal{A}$ is a set of inference expressions, solve returns a bag, and θ is a substitution (most general unifier). The input set of axioms $\mathcal{A}$ to solve is built as the positive

[3]This definition has been simplified since no metapredicates such as negation is considered in this work.

[4]Assuming that john lost his umbrella and it is raining.

information of a set of inference expressions (i.e., axioms $\Delta \vdash A$ such that there is no $\Delta \vdash -A$) plus the restricting axioms with no counterpart positive axioms (i.e., axioms $\Delta \vdash -A$ such that there is no $\Delta \vdash A$).[5] Thus, the meaning of the query $answer(P)$ for Δ_1 is $\{\Delta_1 \vdash answer(ann)\}$ with substitution $\{P/ann\}$ for its single axiom. Observe that a goal such as $own(P, O)$ with respect to Δ_0 gives the meaning $\{\Delta_0 \vdash own(john, umbrella), \Delta_0 \vdash own(ann, umbrella)\}$ because the restricting atom for $john$ is not in Δ_0. The system, for the first goal, returns:

```
DES> -own(john,umbrella) /\ raining => walk(P)
Info: Processing: answer(P) :- Query
{ answer(ann) }
```
$\square$

V. INTEGRATION

This section integrates the two existing different approaches recalled in sections III and IV, including the new features given in Section II. While the first approach considers inference sequences, the second one considers inference expressions. In the following, the first approach is followed to include augmenting databases along inference steps.

The syntax of the Fuzzy Hypothetical Datalog language (FHDL for short) resulting from the integration is the same as the language in Section III but augmented with goals of the form $R_1 \wedge \ldots \wedge R_n \Rightarrow A$, where R_i are rules and A is an atom.[6] Also, rules and atoms in a database can be either positive or restricting in the sense of Section IV.

Because assumptions can be on both rules and fuzzy equations, the notions of both programs and fuzzy relations as defined in Section III become dynamic as opposed to static. Thus, from here on, a database is a collection of both predicates and fuzzy relations which can change along inferencing.

A. Extending FuzzyDES Operational Semantics

From Section IV, adding the embedded implication to a logic language requires to explicitly handle the current database. In addition, from previous Section II, another requirement is the explicit handling of fuzzy relations. So, the operational semantics [15] recalled in Section III is extended to include a changing database Δ. In addition to rules, Δ is extended with fuzzy equations defining both default and user-defined fuzzy relations. A transition system operating on states is defined next.

Let Δ be a FuzzyDES database, including a default similarity relation $\mathcal{R}$ (and possibly others) on the first order alphabet induced by the rules in Δ, where each fuzzy relation has a corresponding fixed t-norm, and let λ be a λ-cut. Let E be a set of tuples $\langle G, \Delta, \theta, \alpha\rangle$ (goal, database, substitution, approximation degree), each one representing a *state* of a computation. Let $\Rightarrow_{HWSLD} \subseteq (E \times E)$ be the transition relation as defined next in Definition 5.1. A *successful inference sequence* for a given state is a sequence ending in the state $\langle \square, \Delta', \theta', \alpha'\rangle$ for a given database $\Delta \subseteq \Delta'$ and some substitution θ' and approximation degree α', where $\square$ represents an empty clause.

[5]This is an adaptation from [22] eliding the definition of closed world assumption for a set of inference expressions because negation is not considered.

[6]This is not a restriction with respect to the syntax in Section IV since $R_1 \wedge \ldots \wedge R_n \Rightarrow \phi$ (with ϕ a conjunctive goal) is operationally equivalent to $R_1 \wedge \ldots \wedge R_n \Rightarrow A$ and $A \leftarrow \phi$.

The symbol $\subseteq$, when applied to databases Δ_1 and Δ_2, stands for a classical subset of graded rules, while for fuzzy equations means that an equation $\mathcal{R}(a, b) = \alpha$ in Δ_1 is contained in Δ_2 whenever $\mathcal{R}(a, b) = \alpha'$ is in Δ_2, for any $\alpha' \in [0, 1]$.

Definition 5.1: Hypothetical Weak SLD (HWSLD) *resolution* is defined as a transition system $\langle E, \Rightarrow_{HWSLD}\rangle$, and whose transition relation $\Rightarrow_{HWSLD}$ is the smallest relation satisfying:

Rule 1: if $R \equiv \langle (A \leftarrow Q); \delta\rangle \ll \Delta$, $\sigma = \mathsf{wmgu}^{\lambda}_{\mathcal{R}}(A, A') \neq fail$, $\lambda \leq \beta = \mathcal{R}(A\sigma, A'\sigma)$, $\mathcal{R}$ is as defined in Δ, and $(\delta \triangle \beta \triangle \alpha) \geq \lambda$:

$$\langle (\leftarrow A' \wedge Q'), \Delta, \theta, \alpha\rangle \Rightarrow_{HWSLD} \langle (\leftarrow Q \wedge Q')\sigma, \Delta, \theta\sigma, \delta \triangle \beta \triangle \alpha\rangle$$

Rule 2: if there exists a successful inference sequence $\langle (\leftarrow A'), \Delta \uplus \Delta_{A'}, \theta, \alpha\rangle \Rightarrow^{*}_{HWSLD} \langle \square, \Delta', \sigma, \alpha'\rangle$:

$$\langle (\leftarrow (\Delta_{A'} \Rightarrow A') \wedge Q), \Delta, \theta, \alpha\rangle \Rightarrow_{HWSLD} \langle (\leftarrow Q)\sigma, \Delta, \theta\sigma, \alpha \triangle \alpha'\rangle$$

where A is either a positive or restricting atom, Q and Q' are conjunctions of atoms, $R \ll \Delta$ represents that R is a standardized apart rule, and $\Delta_1 \uplus \Delta_2$ joins two databases as follows: it returns the union of rules in Δ_i and all fuzzy equations in Δ_i but each $\mathcal{R}(a, b) = \alpha \in \Delta_1$ such that there exists $\mathcal{R}(a, b) = \alpha' \in \Delta_2$ (i.e., $\mathcal{R}(a, b) = \alpha$ is replaced by $\mathcal{R}(a, b) = \alpha'$). An antecedent $d_1 \wedge \cdots \wedge d_n$, with d_i being either a graded rule or fuzzy equation is written as a database $\Delta = \uplus_{1 \leq i \leq n}\{d_i\}$. Without loss of generality, the consequent in the embedded implication is considered to be a single atom.

Example 3: Let us consider the t-norm Gödel and the database Δ:

$$\{\mathcal{R}(a, b) = 0.6,$$
$$\langle r(X) \leftarrow \{\mathcal{R}(p/1, q/1) = 0.7, \langle p(a); 0.8\rangle\} \Rightarrow q(X); 0.9\rangle\}$$

Then, a possible successful inference sequence is:
$\langle \leftarrow r(b), \Delta, \emptyset, 1.0\rangle \Rightarrow_{HWSLD} \langle \{\mathcal{R}(p/1, q/1) = 0.7, \langle p(a); 0.8\rangle\} \Rightarrow q(b), \Delta, \{X/b\}, 1.0 \triangle 0.9\rangle \Rightarrow_{HWSLD} \langle \square, \Delta \uplus \{\mathcal{R}(p/1, q/1) = 0.7, \langle p(a); 0.8\rangle\}, \{X/b\}, 0.6\rangle$.
where the last step with Rule 2 of Definition 5.1 can be performed because there exits the derivation: $\langle \leftarrow q(b), \Delta \uplus \{\mathcal{R}(p/1, q/1) = 0.7, \langle p(a); 0.8\rangle\}, \{X/b\}, 0.9\rangle \Rightarrow_{HWSLD} \langle \square, \Delta \uplus \{\mathcal{R}(p/1, q/1) = 0.7, \langle p(a); 0.8\rangle\}, \{X/b\}, 0.9 \triangle (0.7 \triangle 0.6) \triangle 0.8\rangle$ $\square$

B. Stratified Inference

As introduced in Section IV, stratification enables to give a safe meaning to restricted predicates. Programs in [15] have a fixed stratification for expanded programs, and all of the predicates belong to a single stratum. By adding restricted predicates as required in this current setting (cf. Section II-A), stratification becomes a need, and several strata are expected for a given database. Moreover, since assumptions are involved, the stratification is no longer static because hypotheses in general add new rules and fuzzy equations, therefore modifying the dependency graph and stratification. Thus, we consider the stratification as stated in Section IV, and a stratified inference for fuzzy hypothetical Datalog databases is developed. In what follows, safe databases [15], [25] are required, which in particular help to ensure that answers are closed (ground).

Any inference sequence in a database Δ for a stratum s includes predicates p_i with $str(\Delta, p_i) \leq s$ because, otherwise, the database would not be stratifiable (any arc $p \to q$ implies $str(\Delta, p) \geq str(\Delta, q)$). Next definitions are needed to provide meaning to a database.

Definition 5.2: Given a database Δ and stratum s, its *negative meaning* $[\![\Delta^s]\!]^-$ is the set of graded ground restricting facts $\langle A; \alpha \rangle$ such that $str(\Delta, pred(A)) = s$ and there exists a successful inference sequence

$$\langle \leftarrow A, \Delta, \emptyset, 1 \rangle \Rightarrow^*_{HWSLD} \langle \Box, \Delta, \theta, \alpha \rangle$$

for which there is no other sequence ending in $\langle \Box, \Delta, \theta', \alpha' \rangle$ such that $\alpha' > \alpha$.

Definition 5.3: Given a database Δ and stratum s, its *positive meaning* $[\![\Delta^s]\!]^+$ is the set of graded ground positive atoms $\langle A; \alpha \rangle$ such that $str(\Delta, pred(A)) = s$ and there exists a successful inference sequence

$$\langle \leftarrow A, \Delta, \emptyset, 1 \rangle \Rightarrow^*_{HWSLD} \langle \Box, \Delta, \theta, \beta \rangle$$

for which, first, there is no other sequence ending in $\langle \Box, \Delta, \theta', \beta' \rangle$ such that $\beta' > \beta$ and, second, if $\langle -A; \gamma \rangle \in [\![\Delta^s]\!]^-$, then $\alpha = \beta - \gamma$, and $\alpha = \beta$ otherwise.

Example 4: Let us consider the t-norm Gödel and the propositional database $\Delta = \{\mathcal{R}(p/0, q/0) = 0.9, \langle p; 0.3 \rangle, \langle -p; 0.2 \rangle, \langle q; 0.4 \rangle\}$. Then, there exist the following successful inference sequences:

$\langle \leftarrow -p, \Delta, \emptyset, 1.0 \rangle \Rightarrow_{HWSLD} \langle \Box, \Delta, \emptyset, 1.0 \triangle 0.2 \rangle$ $\qquad \alpha = 0.2$
$\langle \leftarrow -q, \Delta, \emptyset, 1.0 \rangle \Rightarrow_{HWSLD} \langle \Box, \Delta, \emptyset, 1.0 \triangle 0.9 \triangle 0.2 \rangle$ $\qquad \alpha = 0.2$
$\langle \leftarrow p, \Delta, \emptyset, 1.0 \rangle \Rightarrow_{HWSLD} \langle \Box, \Delta, \emptyset, 1.0 \triangle 0.3 \rangle$ $\qquad \alpha = 0.3$
$\langle \leftarrow p, \Delta, \emptyset, 1.0 \rangle \Rightarrow_{HWSLD} \langle \Box, \Delta, \emptyset, 1.0 \triangle 0.9 \triangle 0.4 \rangle$ $\qquad \alpha = 0.4$
$\langle \leftarrow q, \Delta, \emptyset, 1.0 \rangle \Rightarrow_{HWSLD} \langle \Box, \Delta, \emptyset, 1.0 \triangle 0.9 \triangle 0.3 \rangle$ $\qquad \alpha = 0.3$
$\langle \leftarrow q, \Delta, \emptyset, 1.0 \rangle \Rightarrow_{HWSLD} \langle \Box, \Delta, \emptyset, 1.0 \triangle 0.4 \rangle$ $\qquad \alpha = 0.4$

Since both predicates belong to the same stratum ($s = 1$):
$[\![\Delta^1]\!]^- = \{\langle -p; 0.2 \rangle, \langle -q; 0.2 \rangle\}$
$[\![\Delta^1]\!]^+ = \{\langle p; 0.2 \rangle, \langle q; 0.2 \rangle\}$ $\Box$

Definition 5.4: The *meaning* of a database Δ at stratum s is written as: $[\![\Delta^s]\!] = [\![\Delta^s]\!]^+ \cup [\![\Delta^s]\!]^-$.

Definition 5.5: The *unified stratified semantics* of a database Δ with s strata is inductively defined as:

- $\Delta_0 = \emptyset$
- $\Delta_i = [\![\Delta^i]\!] \cup \Delta_{i-1}$, where $1 \leq i \leq s$

Thus, the construction of meanings at a given stratum requires that meanings at lower strata to be already constructed.

Example 5: Let us consider the t-norm Gödel and the propositional database $\Delta = \{\langle q; 0.5 \rangle, \langle r \leftarrow \{\langle p; 0.4 \rangle\} \Rightarrow p \wedge q; 0.6 \rangle\}$. Predicates $p/0$ and $q/0$ are assigned to stratum 1, and $r/0$ to stratum 2. So: $\Delta_1 = \{\langle p; 0.4 \rangle, \langle q; 0.5 \rangle\}$ and $\Delta_2 = \{\langle r; 0.4 \rangle\} \cup \Delta_1$ $\Box$

Therefore, this procedure makes explicit all the facts which can be deduced from the (non-fact) rules of Δ.

C. From FHDL to Hypothetical Datalog

Analogously to FuzzyDES, which translates programs to Datalog, an FHDL program is translated into a non-fuzzy logic language; in this case, into the Hypothetical Datalog version as found in [15]. This benefits from a target language and system that enjoy several advantages (program analysis, automatic transformations, tabled system…), therefore eliding the implementation of a new system. To this end, a translation is defined by extending definitions 4.2 and 4.3 in [15] with restricting rules and facts. The overall idea behind the translation is to make explicit the handling of the weak unification by linearising heads and explicitly computing the approximation degree in the body of each rule and fact. In this work, rules can be either positive or restricting rules. So, Definition 4.2 in [15] is extended as follows:[7] Given a graded rule $\langle [-]p(\overline{t_n}) \leftarrow Q; \delta \rangle$, for each $\mathcal{R}(p, q) = \alpha \in \Delta$ with $\alpha \geq \lambda$, generate the clause:

$$[-]q(\overline{x_n}) \leftarrow (\delta \triangle \alpha) \wedge x_1 \approx t_1 \wedge \cdots \wedge x_n \approx t_n \wedge Q$$

where $\approx$ is the weak unification operator, t_i are terms, x_i are variables, and $\delta \triangle \alpha$ abbreviates the goal $\delta \triangle \alpha \geq \lambda$. This transformation is known as *BPL* expansion. Definition 4.3 in [15] is adapted in a similar way: First, there is only one transformed rule for each original rule in a predicate, and, if there is a fuzzy relation linking to another predicate, only one more rule is added, as follows:

- For each $\langle [-]p(\overline{t_n}) \leftarrow Q; \delta \rangle$, generate the clause:
$$[-]p(\overline{x_n}) \leftarrow \delta \wedge x_1 \approx t_1 \wedge \cdots \wedge x_n \approx t_n \wedge Q$$
- For each $\mathcal{R}(p, q) = \alpha \in \Delta$ with $q \not\equiv p$ and $\alpha \geq \lambda$, generate the clause:
$$[-]q(\overline{x_n}) \leftarrow \alpha \wedge [-]p(\overline{x_n})$$

This transformation is known as *FDES* expansion. In general, *FDES* expansion requires less clauses in the transformation than *BPL* expansion (because several clauses are expected to define a single predicate), therefore saving both space and time along solving. However, when considering the integrated setting which includes restricted predicates, the advantage of the second transformation is not always applicable: some programs become non-stratifiable and therefore rejected.

Example 6: Let us consider the database Δ in Example 4. With the first transformation, Δ is transformed into: $\{(-p \leftarrow 0.2 \triangle 1.0), (-q \leftarrow 0.2 \triangle 0.9)(p \leftarrow 0.4 \triangle 0.9)(p \leftarrow 0.3 \triangle 1.0)(q \leftarrow 0.4 \triangle 1.0)(q \leftarrow 0.3)\}$, its dependency graph is empty and the stratification becomes $\{p/0 : 1, q/0 : 1\}$. In turn, the second transformation yields: $\{(-p \leftarrow 0.2), (p \leftarrow 0.3), (p \leftarrow 0.9 \wedge q), (q \leftarrow 0.4), (q \leftarrow 0.9 \wedge p)\}$, where its dependency graph is $\{p/0 \leftarrow q/0, q/0 \overset{\neg}{\leftarrow} p/0\}$, becoming a non-stratifiable database because of a cycle in the graph including a negative dependency. $\Box$

D. Extending Operational Semantics for Expanded Programs

As in Section V-A, the current database Δ must be explicitly included in the state E with the same four components $\langle G, \Delta, \theta, \alpha \rangle$ (goal, database, substitution, approximation degree) and managed accordingly. In addition, a new inference rule to tackle assumptions is added at the end of the following definition.

Definition 5.6: The *operational semantics for expanded programs* is a transition system $\langle E, \Rightarrow_{Ex} \rangle$ and whose transition relation $\Rightarrow_{Ex}$ is the smallest relation satisfying:

[7] $\overline{o_n}$ denotes the *sequence of syntactic objects* $o_1, \ldots, o_n$ (e.g., $p(\overline{t_n})$ denotes $p(t1, \ldots, t_n)$).

Rule 1: if $\beta \in (0,1]$, and $(\beta \triangle \alpha) \geq \lambda$:

$$\langle(\leftarrow \underline{\beta} \wedge Q'), \Delta, \theta, \alpha\rangle \Rightarrow_{Ex} \langle(\leftarrow Q'), \Delta, \theta, \beta\triangle\alpha\rangle$$

Rule 2: if $\sigma = \mathsf{wmgu}^{\lambda}_{\mathcal{R}}(A,B) \neq fail$, $\lambda \leq \beta = \mathcal{R}(A\sigma, B\sigma)$, $\mathcal{R}$ is as defined in Δ, and $(\beta\triangle\alpha) \geq \lambda$:

$$\langle(\leftarrow \underline{A \approx B} \wedge Q'), \Delta, \theta, \alpha\rangle \Rightarrow_{Ex} \langle(\leftarrow Q'\sigma), \Delta, \theta\sigma, \beta\triangle\alpha\rangle$$

Rule 3: if $([-]p(\overline{x_n}) \leftarrow \beta \wedge x_1 \approx t_1 \wedge \cdots \wedge x_n \approx t_n \wedge Q) \ll \Delta$:

$$\langle(\leftarrow \underline{[-]p(\overline{s_n})} \wedge Q'), \Delta, \theta, \alpha\rangle \Rightarrow_{Ex}$$
$$\langle(\leftarrow \beta \wedge s_1 \approx t_1 \wedge \cdots \wedge s_n \approx t_n \wedge Q \wedge Q'), \Delta, \theta, \alpha\rangle$$

Rule 4: if there exists a successful inference sequence

$$\langle(\leftarrow [-]p(\overline{s_n})), \Delta \uplus \Delta_p, \emptyset, 1\rangle \overset{*}{\Rightarrow}_{Ex} \langle\Box, \Delta', \sigma, \beta\rangle$$

with $\Delta \uplus \Delta_p \subseteq \Delta'$:

$$\langle(\leftarrow \underline{\Delta_p \Rightarrow [-]p(\overline{s_n})} \wedge Q'), \Delta, \theta, \alpha\rangle \Rightarrow_{Ex} \langle(\leftarrow Q'\sigma), \Delta, \theta\sigma, \beta\triangle\alpha\rangle$$

In this system, transition steps are applied to underlined fragments.

E. Stratified Inference for Expanded Programs

Analogously to Section V-B, the next definitions provide the notion of negative and positive meaning for a database of expanded programs which, together, provide the meaning to a database at a given stratum (cf. Definition 5.4).

Definition 5.7: Given a database Δ and stratum s, its *negative meaning* $[\![\Delta^s]\!]^-$ is the set of graded ground restricting facts $\langle -p(t_1, \ldots, t_n)); \alpha\rangle$ such that $str(\Delta, p/n) = s$ and there exists a successful inference sequence $\langle(\leftarrow -p(t_1, \ldots, t_n)), \Delta, \emptyset, 1\rangle \overset{*}{\Rightarrow}_{Ex} \langle\Box, \Delta, \theta, \alpha\rangle$ for which there is no other sequence ending in $\langle\Box, \Delta, \theta', \alpha'\rangle$ such that $\alpha' > \alpha$.

Definition 5.8: Given a database Δ and stratum s, its *positive meaning* $[\![\Delta^s]\!]^+$ is the set of graded ground positive facts $\langle p(t_1, \ldots, t_n); \alpha\rangle$ such that $str(\Delta, p/n) = s$ and there exists a successful inference sequence $\langle\leftarrow p(t_1, \ldots, t_n), \Delta, \emptyset, 1\rangle \overset{*}{\Rightarrow}_{Ex} \langle\Box, \Delta, \theta, \beta\rangle$ for which, first, there is no other sequence ending in $\langle\Box, \Delta, \theta', \beta'\rangle$ such that $\beta' > \beta$ and, second, if $\langle -p(t_1, \ldots, t_n); \gamma\rangle \in [\![\Delta^s]\!]^-$, then $\alpha = \beta - \gamma$, and $\alpha = \beta$ otherwise.

The construction of the complete meaning of a database Δ follows the same procedure as in Definition 5.5 for a unified stratified semantics.

VI. IMPLEMENTATION

The Datalog Educational System (DES) [20] (des.sourceforge.net) is an open-source, Prolog implementation of a deductive database system that, in particular, includes a version of Hypothetical Datalog [21], [22] as recalled in this paper (Section IV), and a system mode for Fuzzy Datalog [15] (Section III) which were incompatible until the development of the current proposal. With respect to [15], this system has been improved to include the expansion of programs in order to include restricted predicates. In addition, since databases are *dynamic* (in the sense that they can grow because of assumptions), both the expansions and $\triangle$-closures are recomputed when needed for each new database context. This also includes the recomputation of the predicate dependency graph and the stratification. As a collateral effect, this dynamic nature leads to a fully interactive fuzzy system, in which graded rules and fuzzy equations can be added and removed from the database, something that was not possible in the version reported in [15], which only allowed for monotonic database changes.

A new on-line system has been developed including the proposal in this paper, which can be accessed at desweb.fdi.ucm.es. Examples in this paper and other databases can be tested there (users with no credentials can log in with a Guest account; recall to use the command `/system_mode fuzzy` to enable fuzzy reasoning).

Next, some distinguished predicates implementing the proposal in this paper are briefly described, and correspond to the published version 6.3 of DES (complete sources can be downloaded from its site).

Rule expansion as described in Section V-C is implemented with the predicate for a clause `Clause` of a predicate p: `expand_rule(+SimDegrees, +Clause, +NVs, +Cin, -Cout, -RuleDegreeVars, -ExpClauses, +IArgs, -IArgsList)` where `SimDegrees` are the approximation degrees α_i for each predicate q such that there exists $\mathcal{R}(p,q) = \alpha_i$; `NVs` are the mapping of names to variables (useful for error reporting and listings); `Cin` (`Cout`, respectively) is the input (output, respectively) constraint store for implementing an efficient weak unification procedure [14]; `RuleDegreeVars` are the variables representing the approximation degrees of the different goals in the clause; and `ExpClauses` are the clauses resulting from the expansion, which depends on the selected expansion (either *BPL* or *FDES* as explained in Section V-C).

This predicate is called whenever a new clause is asserted (via either compiling a program for the first time, or interactively asserting a clause, or asserting a clause when solving an embedded implication). Since fuzzy equations can be retracted and asserted, the predicate `update_fuzzy_expansion(+AssertRetract,+Equation,+CId)` is responsible for updating the database expansion due to either adding or retracting `Equation` (in general, a database contains the default fuzzy relation $\mathcal{R}$ and others) for a given database context `CId` (a positive integer). In turn, the predicate `update_fuzzy_relation(+Rel,+CId)` updates the $\triangle$-closure of the relation `Rel` corresponding to the context `CId`.

Preprocessing in the compilation stage has changed to include the database context identifier because clause expansion needs it (cf. predicate `expand_rule/9`).

The predicate for computing the weak unification of two terms `Term1` and `Term2` [14] has been extended to include the current database context `CId` as follows: `unify(+Term1, +Term2, +CId, +Cin, -Cout, -Degree)`, which in particular calls to the new version for retrieving the unification degree: `unification_degree(+Atomic1, +Atomic2, +CId, ?Block, -Degree)`. One of the clauses of this predicate includes the call `'~'(Atomic1, Atomic2, CId, Block, Degree)` to the predicate `'~'/5`, which explicitly represents the default fuzzy relation $\mathcal{R}$. Each fact in this predicate is extended with the context identifier `CId` for each different database along assumptions. This predicate is generated with `gen_rb(+CId)`, which in turn is called each time $\mathcal{R}$ changes in the database context `CId`.

The predicate `compute_restricted_meaning(+Q,+CId)` implements Definition 5.4 by traversing the extension table, looking for the maximum positive and restricting entries (Definitions 5.7 and 5.8).

The function $memo$ (cf. [15]) implements a form of tabling (following [8]) by adding to the extension table entries of the form `et(Hash, Atom, Source, CId, It)`, where `Hash` is a hash value for quick index-based lookup, `Atom` is an inferred fact A together with its approximation degree α in its last argument (corresponding to $\langle A; \alpha \rangle$), `Source` is the fact source (for duplicates), `CId` is the database context identifier, and `It` is the fixpoint iteration in which the fact has been inferred (not used yet but ready for an implementation of differential optimization [3]). This function is called in a fixpoint computation for each strata (predicate `solve_star(+Query,+Stratum,+CId)`) whenever a new fact is inferred. Each stratum is computed from the lower to the upper with the predicate `solve_stratified(+Query,+CId,-Undefined)` (which can return undefined tuples for non-stratifiable databases). This predicate implements a top-down, bottom-up construction of the fixpoint, focusing on a user query instead of solving the meaning of the whole database, therefore improving efficiency with respect to a simple bottom-up approach.

VII. Conclusion

A novel proposal for a fuzzy hypothetical deductive database has been described. To the best of authors' knowledge, there is neither a similar approach in the existing literature nor a system. Syntax, operational semantics and stratified inference have been developed for committing to a series of requirements in decision-support applications with vague knowledge. As a strong result of this research, a working system implementing this scenario has been made available, which can be tested, downloaded, and its sources can be inspected.

This work can be extended with duplicates, strong constraints, negation, data types, and aggregates in the fuzzy setting, to name a few. Also, compiling can be enhanced by analysing the program and generate a specific transformation for each predicate: apply the *FDES* transformation if it does not break stratification, and *BPL* otherwise. With respect to the system as a whole, it can be improved in a number of ways including a more efficient parser and the implementation of a differential optimization [3].

References

[1] S. Abiteboul, R. Hull, and V. Vianu, Eds., *Foundations of Databases: The Logical Level*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] Á. Achs and A. Kiss, "Fuzzy extension of datalog," *Acta Cybernetica*, vol. 12, no. 2, pp. 153–166, 1995.

[3] M. Alvarez-Picallo, A. Eyers-Taylor, M. Peyton Jones, and C.-H. L. Ong, "Fixing incremental computation," in *Programming Languages and Systems*, Cham: Springer International Publishing, 2019, pp. 525–552.

[4] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn, "Design and implementation of the logicblox system," in *Proceedings of the 2015 ACM SIGMOD*: ACM, 2015, pp. 1371–1382.

[5] P. Avgustinov, O. de Moor, M. P. Jones, and M. Schäfer, "QL: object-oriented queries on relational data," in *ECOOP*, ser. LIPIcs, vol. 56. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 2:1–2:25.

[6] A. J. Bonner, "Hypothetical Datalog: Complexity and Expressibility," *Theoretical Computer Science*, vol. 76, pp. 3–51, 1990.

[7] A. J. Bonner and L. T. McCarty, "Adding negation-as-failure to intuitionistic logic programming," in *Proc. of the North American Conference on Logic Programming*, The MIT Press, 1990, pp. 681–703.

[8] S. W. Dietrich, "Extension tables: Memo relations in logic programming," in *IEEE Symp. on Logic Programming*, 1987, pp. 264–272.

[9] D. Easley and J. Kleinberg, *Graph Theory and Social Networks*. Cambridge University Press, 2010, pp. 19–20.

[10] S. Greco, C. Molinaro, I. Trubitsyna, and E. Zumpano, "NP datalog: A logic language for expressing search and optimization problems," *TPLP*, vol. 10, no. 2, pp. 125–166, 2010.

[11] T. J. Green, "LogiQL: A Declarative Language for Enterprise Applications," in *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on PODS*: ACM, 2015, pp. 59–64.

[12] S. Guadarrama, S. Muñoz, and C. Vaucheret, "Fuzzy Prolog: A new approach using soft constraints propagation," *Fuzzy Sets and Systems, Elsevier*, vol. 144, no. 1, pp. 127–150, 2004.

[13] P. Julián-Iranzo and C. Rubio-Manzano, "A sound and complete semantics for a similarity-based logic programming language," *Fuzzy Sets and Systems*, pp. 1–26, 2017.

[14] P. Julián-Iranzo and F. Sáenz-Pérez, "An efficient proximity-based unification algorithm," in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, July 2018, pp. 441–448.

[15] P. Julián-Iranzo and F. Sáenz-Pérez, "A fuzzy datalog deductive database system," *IEEE Transactions on Fuzzy Systems*, vol. PP, no. 99, pp. 1–1, 2018.

[16] R. Kowalski and D. Kuehner, "Linear resolution with selection function," *Artificial Intelligence*, vol. 2, 1971.

[17] J. Medina, M. Ojeda-Aciego, and P. Vojtáš, "Similarity-based unification: a multi-adjoint approach," *Fuzzy Sets and Systems*, vol. 146, no. 1, pp. 43–62, 2004.

[18] J. M. Medina, O. Pons, J. C. Cubero, and M. A. Vila, "FREDDI: A fuzzy relational deductive database interface," *International Journal of Intelligent Systems*, vol. 12, no. 8, pp. 597–613, 1997.

[19] C. Rubio-Manzano and P. Julián-Iranzo, "Fuzzy Linguistic Prolog and its Applications," *Journal of Intelligent and Fuzzy Systems*, vol. 26, pp. 1503–1516, 2014.

[20] F. Sáenz-Pérez, "DES: A Deductive Database System," *Electronic Notes on Theoretical Computer Science*, vol. 271, pp. 63–78, March 2011.

[21] F. Sáenz-Pérez, "Implementing tabled hypothetical datalog," in *Proceedings of the 25th IEEE ICTAI*, November 2013.

[22] ——, "Restricted predicates for hypothetical datalog," *Electronic Proceedings in Theoretical Computer Science*, vol. 200, no. 0, pp. 64–79, 2015.

[23] M. I. Sessa, "Approximate reasoning by similarity-based SLD resolution." *Theoretical Computer Science*, vol. 275, no. 1-2, pp. 389–426, 2002.

[24] G. Terracina, N. Leone, V. Lio, and C. Panetta, "Experimenting with recursive queries in database and logic programming systems," *TPLP*, vol. 8, no. 2, pp. 129–165, 2008.

[25] J. D. Ullman, *Database and Knowledge-Base Systems, Vols. I (Classical Database Systems) and II (The New Technologies)*. Computer Science Press, 1988.

[26] P. Vojtáš, "Fuzzy Logic Programming," *Fuzzy Sets and Systems*, vol. 124, no. 1, pp. 361–370, 2001.

[27] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, "SWI-Prolog," *Theory and Practice of Logic Programming*, vol. 12, no. 1-2, pp. 67–96, 2012.