electronics

MDPI

Article

GEAR: A General Inference Engine for Automated MultiStrategy Reasoning

Stefano Ferilli

Department of Computer Science, University of Bari Aldo Moro, 70125 Bari, Italy; stefano.ferilli@uniba.it

Abstract: The pervasive use of AI today caused an urgent need for human-compliant AI approaches and solutions that can explain their behavior and decisions in human-understandable terms, especially in critical domains, so as to enforce trustworthiness and support accountability. The symbolic/logic approach to AI supports this need because it aims at reproducing human reasoning mechanisms. While much research has been carried out on single inference strategies, an overall approach to combine them is still missing. This paper claims the need for a new overall approach that merges all the single strategies, named *MultiStrategy Reasoning*. Based on an analysis of research on automated inference in AI, it selects a suitable setting for this approach, reviews the most promising approaches proposed for single inference strategies, and proposes a possible combination of deduction, abduction, abstraction, induction, argumentation, uncertainty and analogy. It also introduces the GEAR (General Engine for Automated Reasoning) inference engine, that has been developed to implement this vision.

Keywords: automated reasoning; multistrategy inference; deduction; abstraction; abduction; argumentation; induction; uncertainty; analogy

check for updates

Citation: Ferilli, S. GEAR: A General Inference Engine for Automated MultiStrategy Reasoning. *Electronics* **2023**, *12*, 256. https://doi.org/10.3390/electronics12020256

Academic Editor: Juan M. Corchado

Received: 26 October 2022
Revised: 28 December 2022
Accepted: 30 December 2022
Published: 4 January 2023

1. Introduction

Research in Artificial Intelligence (AI) has followed two, different but complementary, directions: the Logical/Symbolic/Neat approach vs. the Analogical/Connectionist/Scruffy one [1,2]. The former [3], based on numerical/statistical techniques, resulted in systems that are efficient and noise-tolerant, but that cannot capture the complex network of relationships existing in real-world situations, and are not understandable by humans (so-called 'black box'). The issue is well-known and faced under the name eXplainable Artificial Intelligence (XAI) [4,5], also among big players, such as The Royal Society [6], IBM [7], and DARPA [8]. The other approach [9], based on symbolic/logic techniques, and specifically on the First-Order Logic (FOL) setting, can natively handle multi-relational representations and reproduce high-level human reasoning mechanisms. Both approaches are needed to handle the many different aspects involved in carrying out 'intelligent' behavior in the real world. The former is more suitable to reproduce perception and sub-conscious mechanisms, while the latter is more appropriate to implement conscious reasoning [1].

With the progressively pervasive use of AI today, many tasks are being addressed that require human-compliant AI approaches and solutions, especially in critical domains, so as to enforce trustworthiness and support accountability of the automated systems. A fundamental component in this landscape is the ability to explain the system's decisions in human-understandable terms [10]. While in XAI many attempts are ongoing to superimpose explanations onto subsymbolic, black-box AI models, symbolic techniques can provide native transparency and explanation capabilities. Hence, the motivation for this paper to focus on AI approaches is based on logical inference.

The most classical, investigated, and for some respects obvious, kind of inference to be reproduced is deduction. Still, it is a very strict kind of inference, that can deduce

absolute truths only from certain and complete knowledge. The complexity of the real world requires humans to use several other kinds of inference to tackle issues such as missing or wrong information, uncertainty, efficiency, etc. As a consequence, also the AI literature investigated many inference strategies, proposing automated procedures that can simulate the way in which they are carred out by humans . Single strategies have been typically studied separately, resulting in the definition and implementation of different inferential operators. Unfortunately, several studies demonstrated that single inference approaches suffer from significant limitations, that can be overcome only by combining many different inference strategies together, so as to leverage the advantages of each and compensate for each other's shortcomings [11], just like humans do. However, so far the mainstream literature on automated reasoning only focused on the combination of very small sets (most often just pairs) of inference strategies and operators. This provides the second motivation for this paper.

So, the objectives of this paper are:

- To propose a research direction aimed at combining as many strategies as possible in one integrated framework;
- To identify the most promising setting in logic-based AI for hosting the widest possible combination of different inference strategies;
- To identify the single approaches to automated inference strategies proposed in the literature for such a setting that are compliant to each other, and amenable to a smooth integration; and
- To provide a practical example of working the combined framework.

A systematic literature review was carried out, with Google and DuckDuckGo as search engines. We performed several searches using different strings, aimed at identifying relevant literature starting from the general topic and progressively focusing on more specific strategies and related approaches. First, we looked for "multistrategy reasoning", and found that it does not exist as a research branch of AI. Thus, we tried with more explicit search strings, such as "reasoning frameworks in artificial intelligence" and "combination of inference strategies in artificial intelligence", but only very general and non-technical pages were found as results. We then started narrowing the focus to logic, using strings such as "logic-based approaches to automated reasoning" or "logic-based reasoning frameworks in artificial intelligence": here, accessing most relevant pages in the results and following the links they contained, we could realize that Logic Programming was the setting in which most investigation on single strategies were carried out, and that Machine Learning was the branch in which most attempts at combining different strategies were made. Next, we performed searches on the single strategies, in these settings, using strings such as "deduction, abduction, ... techniques in artificial intelligence" and "deduction, abduction, ... in logic programming". Here, we mostly found the literature we were already aware of, with most recent contributions dating back to a few years ago. Finally, to further check whether new works were not found with these searches, we tried searching the titles of the papers we already knew from our previous research, to check whether and which new literature was citing them and extending their results. The most recent works we found are those cited in the next sections. After such a review we may conclude that, to the best of our knowledge, this is the first proposal and attempt to define and implement an overall solution combining in a single framework many approaches implementing different automated reasoning strategies.

Given the above landscape and motivations, this paper contributes from three perspectives:

Position: It posits the need for symbolic reasoning and for an overall framework combining several different inference strategies (in principle, as many as possible). Such a framework would mimic the variety and flexibility of human reasoning. Of course, an ultimate solution to this problem is beyond the scope of this paper and of the current state of the art in AI; still, this paper is an attempt to define the problem and provide a partial solution, as a starting point to be further expanded in future research to try and approximate more and more closely human reasoning.

Survey: Based on this position, its largest contribution is the identification of a suitable logic setting on which basing the framework, and a summarization of the literature on automated reasoning within the identified setting. Specifically, this paper proposes a two-fold survey of the current research landscape: first, it overviews the approaches developed for single inference strategies; then, it overviews the existing attempts at combining those strategies.

Proposal: The limitations of these attempts motivate the novel contribution, which consists in proposing a new research direction, named *Multistrategy Reasoning* (MSR), specifically aimed at investigating approaches that combine together as many inference strategies as possible in a single, integrated framework. We also introduce the GEAR inference engine as our proposed implementation for MSR, and the first practical contribution to this endeavor. It combines many compatible approaches described in the survey that were developed separately and are so far combined in a limited way.

We selected Logic Programming (LP) as the richest and most promising setting in which this view can be achieved, for several reasons. First, LP is the fragment of logics traditionally connected to computer implementation: logic (Prolog) programs have the same power as standard algorithmic programs, but exploit logic deduction as the mechanism for executing programs described as sets of logic formulas. Second , many different automated inference strategies and operators investigated in the literature adopt this setting, which is a pre-requisite for their tight integration. Within the LP-related literature, we reviewed and selected the works that we considered more appropriate for a smooth integration, covering more than 25 years of research on this topic. A first criterion for selection was that they all spread from, and extend, the basic LP setting, that provides deductive inference. Then, among the various proposals based on this setting, the second criterion was to select those that can more smoothly and immediately be combined, since their assumptions and mechanisms are compliant with each other.

The paper is organized as follows. After recalling the basics of inference strategies and of the formalism we will adopt, selected approaches proposed in the literature for the single strategies of interest are described. Then, the combinations of these strategies proposed in previous works are discussed from a new perspective that highlights how they can be merged in an overall framework, finally introducing the GEAR system that implements this cooperation, along with its formalism. The last section concludes the paper and outlines future work issues.

2. Background

We start by recalling a historical theoretical framework for the combination of different inference strategies, and by introducing the formalism we will use for our proposal.

2.1. Basics

A theoretical framework is useful as a starting point to identify the role and status of different inference strategies, albeit developed from the specific perspective of learning agents, which is the *Inferential Theory of Learning* (ILT) [11]. Learning is usually defined as a change in behavior due to experience. ILT identifies three fundamental factors affecting a 'learning task': the *input* (i.e., the information that is provided to the learner), the *background knowledge* (BK) (i.e., what the learner already knows from previous experience), and the *goal* (i.e., what the learner wants to accomplish). The learning process can be characterized in terms of *knowledge transmutations*, i.e., inference patterns associated to different reasoning methods that produce new knowledge based on the input and the BK. Some knowledge transmutations are performed explicitly and some others implicitly, depending on the knowledge representation and computational mechanism they use. They are explicit in symbolic systems, implicit in subsymbolic ones.

Each knowledge transmutation is characterized by the type of inference on which it is based. The *fundamental equation for inference*:

$$P \cup BK \models C \tag{1}$$

where P is a set of statements (called *premise*), BK is the reasoner's *background knowledge*, and C is a set of statements (called *consequent*) allows us to define different types of inference independently of the representation language used. A first relevant distinction is between deductive and inductive inference. The former derives C given P and BK, thus traversing 'forward' Equation (1), and is truth-preserving (if the premises are true, the conclusion must be true). The latter hypothesizes P given C and BK, traversing it 'backward', and is falsity-preserving (if the conclusion is false, the premise cannot be true). Another distinction is between *universal* inferences, that interpret Equation (1) as a 'strong' (valid) entailment, and *contingent* ones, interpreting it in a 'weak' (approximate, common-sense) way. Combining these two dimensions, one obtains universal deduction (which is strictly truth-preserving), universal induction (which is strictly falsity preserving), contingent deduction and contingent induction (or *abduction*) (which are based on domain-dependent relationships and are truth- and falsity-preserving , respectively, only to the extent if the contingent dependencies involved in reasoning are true). They differ only if the entailment in Equation (1) involves causality (i.e., P is the cause of consequence C), in which case contingent deduction produces possible consequences of the given causes, and abduction produces possible causes of the given consequences. At the intersection of the two dimensions is *analogy*. This inference is based on equivalence and can be considered as a combination of induction and deduction, where the former is used to detect an analogical matching (i.e., the properties and/or relations that play an analogous role in two domains) and the latter proposes unknown knowledge in the target domain based on the hypothesized analogical matching.

2.2. Logic Programming Framework

As said, our proposal is based on the Logic Programming (LP) framework [12]. It is the fragment of FOL, in which executable computer programs can be written as logic theories, and for which several kinds of inference strategies have been investigated. LP is based on clausal logic. This is the portion of FOL working on *clauses*, i.e., logical expressions of the form

$$l'_1; \ldots; l'_m \leftarrow l_1, \ldots, l_n$$

denoting implications

$$l_1 \wedge \ldots \wedge l_n \Rightarrow l'_1 \vee \ldots \vee l'_m$$

where the l'_j's ($j = 1, \ldots, m$) are *atoms*, and an atom is a predicate applied to terms as arguments (a predicate p requiring k arguments is usually denoted as p/k). In turn, a term is a constant, a variable, or an n-ary function recursively applied to n terms as arguments. The l_i's ($i = 1, \ldots, n$) are *literals*, i.e., atoms or negated atoms (the negation of an atom a will be denoted, in the following, by $\neg a$ or by $\overline{a}$). An atom, literal or clause is *ground* if it does not involve any variable. Given a clause C, $terms(C)$ denotes the set of terms occurring in C. $l'_1; \ldots; l'_m$ (the conclusion of the implication) is called the *head* of the clause, and $l_1, \ldots, l_n$ (the premise of the implication) is called the *body* of the clause (the set of literals in the body of C is denoted by $body(C)$). More details about clausal representation and LP can be found in [12,13].

In particular, logic programs (or 'theories') consist of *Horn clauses* $l_0 \leftarrow l_1, \ldots, l_n$, where the head may include at most one atom . A *rule* is a (Horn) clause having both the head and the body; a Horn clause having only the head (l_0) is a *fact*; one having only the body ($\leftarrow l_1, \ldots, l_n$) is a *goal*; the *empty clause*, having no head nor body and denoted by $\square$, represents a contradiction.

Definition 1 ([14]). *A clause is* domain restricted *if all variables occurring in its body also occur in its head; it is* range restricted *iff all variables occurring in its head also occur in its body.*

Definition 2 ([15]). *A Horn clause is* linked *if all of its literals are; a literal is linked if at least one of its arguments is; an argument of a literal is linked if the literal is the head of the clause or if another argument of the same literal is linked.*

More specifically, we consider *Datalog* clauses. The Datalog language [16], developed for deductive databases, restricts LP so that only variables and constants (i.e., no function symbols) are allowed as terms. It is standard practice in the community to work in the Datalog fragment, because it allows direct connection to relational DBs for handling the knowledge efficiently . We can focus on Datalog clauses without loss of generality because any clause can be translated into a Datalog one, and vice-versa, using the *flattening* and *unflattening* procedures [17]. Moreover, without loss of generality [18] , we extend the *unique names assumption* [19] typically adopted for constants, by the *Object Identity* assumption [18], requiring that "*in a clause, terms* (including variables) *denoted by different symbols must be distinct*", i.e., they must denote different entities.

Clauses can be compared to each other based on a generality ordering. The classic generality ordering used in logic is the implication, according to which a formula is more general than another if any model for the former is also a model for the latter, i.e., whenever the former is true, the latter is too. Since it is undecidable, various attempts were made in the literature to define weaker orderings that are computationally viable . The most famous is θ-subsumption:

Definition 3 ([12]). *Given two clauses C and D, D θ-*subsumes *C if $\exists\theta$ substitution s.t. $D\theta \subseteq C$. Where a* substitution *θ is a function replacing variables with terms. Applying a substitution θ to a logic formula F, denoted as $F\theta$, means replacing each variable in F with the corresponding term in θ, if any. A* grounding substitution *replaces variables with constants; a grounding of a clause C is the application of a grounding substitution to C.*

3. Inference Strategies

Let us now provide state-of-the-art definitions and settings for a number of inference strategies that can serve several needs of an automated reasoning system.

3.1. Deduction

Deduction is the kind of inference aimed at making explicit knowledge that is only implicit in the available knowledge, but is a strict consequence thereof.

One way to carry out deductive inference in LP is based on (a specific version of) *resolution*. It is an inference rule by which, given two clauses

$$C' : l'_0 \leftarrow l'_1, \ldots, l'_n \qquad\qquad C'' : l''_0 \leftarrow l''_1, \ldots, l''_m$$

and a substitution θ such that $l'_i\theta = l''_0\theta$ for some $i \in \{1, \ldots, n\}$, then the clause

$$C : (l'_0 \leftarrow l'_1, \ldots, l'_{i-1}, l''_1, \ldots, l''_m, l'_{i+1}, \ldots, l'_n)\theta$$

obtained by "*resolving C' on* literal l'_i", is a logical consequence of C' and C''. Thanks to a Subsumption Theorem provided in [13], this allows us to define deduction based on repeated resolution steps (called a *derivation* and denoted by $\vdash$):

Definition 4 ([14]). *Let Σ be a set of clauses, and C a clause. We say that C can be* deduced *from Σ if C is a tautology, or there exists a clause D s.t. $\Sigma \vdash D$ and D θ-subsumes C.*

In traditional LP, a conjunction of literals $l_1 \wedge \ldots \wedge l_n$ ($n > 0$) can be proven by *refutation*: if adding its negation (which is a goal $\leftarrow l_1, \ldots, l_n$) to a program P the empty clause (i.e., a contradiction) can be obtained by repeated application of resolution steps, then the

conjunction is proven in P. Thus, this is a kind of *backward* (or goal-driven) strategy, in which deduction is focused on proving a given goal. In the opposite perspective, given a set of facts a *forward* (or data-driven) strategy picks in turn all the available facts and resolves them with the clause bodies whenever possible, progressively deriving all of their possible consequences .

If the body of clauses may contain negated literals, the resulting programs are called *general* logic programs. In the backward approach, such negated literals are proved using the *Negation as Failure* (NAF) rule [20]: if an atom cannot be proven by refutation, then its negation is assumed to be true. The NAF rule stems from the Closed World Assumption: only what is reported in the program is true; whatever is not reported in the program is considered as false.

3.2. Abstraction

Abstraction reduces the amount of information conveyed by a set of facts, called the *reference set* [11]. This reduces the computational load needed to process the set of facts, provided that the information that is relevant to the achievement of a goal is preserved.

According to [21], it is based on a *reasoning context* $\langle P(W), S, L \rangle$. A world W contains various kinds of (atomic or compound) objects, each kind with specific properties. $P(W)$, the world-perception, is a perception system consisting of a set of sensors, each dedicated to a specific signal and with a resolution that determines the minimum difference it can distinguish between two signals. S, the structure, stores the information sensed by $P(W)$ as a set of tables in a relational database, in which each object is associated with a unique identifier. To intensionally describe the world, a logic language L associates tables in S with predicate and function symbols, on which reasoning can be carried out. $P(W)$ can be formalized as a 4-tuple $P(W) = (O, A, F, R)$, where O is a set of perceived objects, A is a set of object attributes, F is a set of functions, and R is a set of relations. These are the same items that are considered when building the conceptualization of a world to formally describe it.

A set of values provided by the sensors in P is called a *signal pattern* or *configuration*; the set of all possible configurations that $P(W)$ can distinguish is denoted by Γ. Different perceptions of the same world may be obtained (e.g., due to the kind or resolution of the sensors, the focus or perspective, etc.). In ML, this corresponds to the phase of *feature selection*. A perception system is said to be *simpler* than another if it hides some information that is apparent in the other, thus offering less information to manipulate. More formally:

Definition 5 ([21]). *Given a world W, and $P_1(W), P_2(W)$ two perception systems for W, with configuration sets Γ_1, Γ_2 (resp.), $P_2(W)$ is* simpler *than $P_1(W)$ if $\exists f : \Gamma_1 \to \Gamma_2$ injective but not bijective*

1. *$\exists f : \Gamma_1 \to \Gamma_2$ injective function (every $\gamma_1 \in \Gamma_1$ uniquely determines a $\gamma_2 \in \Gamma_2$)*
2. *the inverse function f^{-1} does not exist (some γ_1 cannot be uniquely reconstructed from γ_2)*

An abstraction switches from a perception system to a simpler one:

Definition 6 ([21]). *Let W be a world, and $R_g = (P_g(W), S_g, L_g)$, $R_a = (P_a(W), S_a, L_a)$ two reasoning contexts, where P_a is simpler than P_g. An* abstraction *is a mapping $A : P_g(W) \to P_a(W)$.*

In this definition, subscript g stands for 'ground' and subscript a stands for 'abstract'. Note that an abstract configuration is uniquely determined from a ground one, but the opposite (an operation called *concretion*), is unfeasible unless additional information is provided. Thus, abstraction takes place at level $P(W)$, that is the source of information, and then it propagates to the other levels in the reasoning context only as a side-effect, in the following order: $P(W)$ is recorded in S and then described by L. In this view , changes in the structure and language are mere consequences of different mappings in the world perception.

Abstraction happens by means of a set of operators. This set includes general operators that allow to: group indistinguishable objects into equivalence classes; group a set of objects in the ground world to form a new compound object that replaces them in the abstract world; ignore terms in the abstract world, where they disappear; merge a subset of values that are considered indistinguishable; drop a subset of arguments, thus reducing the arity or a relation; eliminate *all* arguments in a function or relation, so that it moves from a predicate logic to a propositional logic setting at the language level (*propositional abstraction*). Domain-specific operators can also be included. Each level in the reasoning context has its own set of operators: Ω for $P(W)$, Σ for S, and Λ for L. Operators in each of these sets correspond to the operators in the others, so that, given a ground reasoning context, from Ω one can obtain not only the perceived world abstraction P_a, but also its abstract structure S_a and language L_a, by applying the corresponding operators in Σ and Λ, respectively.

At the language level, abstraction is defined as a mapping between representations that are related to the same reference set but contain less detail, so as to preserve some properties and discard others. It aims at helping to solve a problem in the ground description and at making the search for a solution more easily manageable [22].

Definition 7 (adapted from [22]). *Given two clausal theories T and T' built upon different languages $\mathcal{L}$ and $\mathcal{L}'$ (and derivation rules), an* abstraction *is a tuple (T, T', f), where f is a computable total mapping between clauses in $\mathcal{L}$ and those in $\mathcal{L}'$. We will call T the* ground theory *and T' the* abstract theory.

Abstraction is truth-preserving, since it is based on deduction.

3.3. Abduction

Abduction is devoted to cope with missing information, by guessing unknown facts when they are needed for a given purpose.

The Abductive Logic Programming (ALP) [23–25] framework extends LP by allowing to guess some 'abducible' facts (abductive hypotheses) that are not stated in the available knowledge but are needed to solve a given problem. The problems can be observations to be explained (as in classical abduction) or goals to be achieved (as in standard LP). Of course, there may be many plausible explanations for a given observation, and thus abductive explanations are not conclusive, requiring strategies to filter and rank explanations.

Definition 8 ([25]). *An* abductive logic program *(or* abductive theory*) consists of a triple $\langle P, A, I \rangle$, where:*

P is a general logic program;

A (Abducible predicates) is a set of predicates;

I (Integrity Constraints) is a set of formulas that must be satisfied by the abductive hypotheses.

In principle, the specific atoms on which abductions can be made should be the 'abducibles' [23]. By extension, using an *abducible predicate* is a quick way to say that any atom built on that predicate can be abduced. In other words, an abducible predicate is a kind of claim that may be abduced, while an abducible literal is a specific claim that may be abduced

Definition 9 (Abductive explanation [25]). *Given an abductive theory $T = \langle P, A, I \rangle$ and a formula G, an* abductive explanation *Δ for G is a set of ground atoms of predicates in A s.t. $P \cup \Delta \models G$ (Δ explains G) and $P \cup \Delta \models I$ (Δ is consistent).*

Among the various procedures proposed in the literature to obtain abductive explanations for abductive logic programs, also when negated literals are used in the body [26], we adopt the one proposed by [27]. It interleaves two phases: *abductive* and *consistency derivations*. An *abductive derivation* is the standard LP derivation extended in order to

consider abducibles. When an unknown abducible literal δ must be proved, it is added to the current set of abductive hypotheses , starting a *consistency derivation* to check that no integrity constraints involving δ is violated. In doing this, the *abductive derivation* is used to solve each goal, which might further extend the set of abductive hypotheses . The procedure returns a minimal abductive explanation (according to set inclusion) if any, otherwise it fails.

Traditional ALP considered Integrity Constraints (ICs) in the form of LP goals ($\leftarrow l_1, \ldots, l_n, \neg l_{n+1}, \ldots, \neg l_{n+m}$), i.e., negations of conjunctions of literals (the *nand* logistic operator). Reference [28] extended traditional ALP into the *Expressive ALP* (EALP) framework, allowing additional types of ICs based on other operators, and specifically:

and all listed literals must be true;

nor all listed literals must be false;

xor exactly one of the listed literals must be true;

iff the *and* constraints on the first and second list of literals must either both succeed or both fail;

if based on the $X \Rightarrow Y \equiv \neg X \vee Y$ equivalence: the *nand* constraint on the list of literals in the premise or the *and* constraint on the list of literls in the consequence must be true;

or at least one of the listed literals must be true;

nand at least one of the listed literals must be false.

This set of typed ICs strictly extends the expressiveness of the framework, because they cannot be simulated by the traditional ICs used in ALP. To handle them, some changes are required in the abductive procedure, and specifically in the consistency derivation [28].

3.4. Uncertain Reasoning

Much research investigated how to combine logical and statistical inference, so that the former supports high-level reasoning strategies, and the latter improves flexibility and robustness. From an LP perspective, they resulted in the *Probabilistic Logic Programming* (PLP) setting [29].

Very relevant in PLP is the distribution semantics [30], by which a probabilistic logic program defines a probability distribution over a set of normal logic programs (called *worlds*). Examples of languages based on the distribution semantics are ProbLog [31], PRISM [32], LPADs [33], and CP-Logic [34]. They differ in the way they define the distribution. Both ProbLog and PRISM allow to set probabilities only on facts; the former allows two alternatives only (true or false). LPADs and CP-Logic offer a more general syntax than ProbLog and PRISM. Since in CP-Logic some programs to which a causal meaning cannot be attached are not valid; in the following, we will consider LPADs.

A Logic Program with Annotated Disjunctions (LPAD) consists of a finite set of disjunctive clauses in which each atom in the head is annotated with a probability, of the form:

$$C = h_1 : \pi_1; \ldots; h_n : \pi_n; null : (1 - \sum_{k=1}^{n} \pi_k) :- b_1, \ldots, b_m.$$

where the semicolon is the logical disjunction operator, $h_1, \ldots h_n$ are atoms, $b_1, \ldots, b_m$ are literals, $\pi_1, \ldots, \pi_n \in]0, 1]$ are such that $\sum_{k=1}^{n} \pi_k \leq 1$, and *null* (representing the case that none of the h_i's is true) does not appear in the body of any clause. Note that, if $n = 1$ and $\pi_1 = 1$, C is a non-disjunctive clause. Without loss of expressive power [29], probabilistic facts are considered as independent.

Concerning the distribution semantics for Datalog programs , an *atomic choice* (C, θ_j, k) is a selection of $h_k\theta_j$ from $C\theta_j$, where θ_j is a grounding substitution and $k \in \{1, \ldots, n\}$. It represents an equation of the form $X_j = k$ where X_j is a random variable associated with $C\theta_j$. A *composite choice* κ is a *consistent* set of atomic choices for the clauses in a program, meaning that the choices do not select different heads for a given ground clause. The probability of a composite choice κ is $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \pi_{ik}$. A total composite choice,

or *selection*, σ includes one atomic choice for every grounding of each probabilistic clause. It identifies a *world* (a logic program) w_σ, whose probability is $P(w_\sigma) = P(\sigma)$. Since we work in Datalog (i.e., the program does not contain function symbols), the set of worlds $W_T = \{w_1, \ldots, w_m\}$ is finite and $P(w)$ is a distribution over worlds: $\sum_{w \in W_T} P(w) = 1$.

The conditional probability of a query (ground atom) q given a world w is defined as: $P(q \mid w) = 1$ if q is true in w ($w \models q$), or 0 otherwise. By marginalization, the probability that query q is true is:

$$P(q) = \sum_w P(q, w) = \sum_w P(q \mid w) P(w) = \sum_{w \models q} P(w).$$

It is worth concluding this section by mentioning a completely different approach to uncertainty, not based on a formal definition of probabilities. In fact, the mathematical theory of probability requires complex computations that are not actually carried out by humans when dealing with uncertainty in everyday reasoning . Rather, they use informal but quick ways of estimating the certainty of the information they handle. The most relevant approach aimed at simulating this behavior is perhaps the one implemented in MYCIN, one of the first successful expert systems available in the literature [35]. It defines a certainty function $c(\cdot) \in [-1, +1]$ and, inspired by fuzzy logic [36]; it computes the composition of certainties as follows.

Given two facts F_1 and F_2, the certainty of their composition using the basic logistic operators is:

- $c(F_1 \wedge F_2) = \min(c(F_1), c(F_2))$ (since both pieces of information are required, the less certain affects the overall certainty);
- $c(F_1 \vee F_2) = \max(c(F_1), c(F_2))$ (since the two pieces of information are interchangeable, the certainty of the most certain can be assumed);
- $c(\neg F_1) = 1 - c(F_1)$ (the certainty of a negation is the complement of the original certainty).

3.5. Argumentation

Argumentation is the inferential strategy aimed at dealing with inconsistent knowledge, in order to distinguish which of several contrasting positions in a dispute are justified. In a dispute, the participants make claims (the *arguments*) to support their own position, to attack the arguments for competing positions of the other participants, and to defend their position from the attacks of the others. Abstract argumentation, in particular, stemmed from ALP and focuses only on the inter-relationships among the arguments, neglecting their internal structure or interpretation.

Abstract Argumentation Frameworks (AFs) can be graphically represented as graphs, whose nodes are the arguments and whose arcs represent the relationships between pairs of arguments. As originally defined [37], they can express only attacks among arguments. Several lines of research introduced additional features in the AFs, usually studied in isolation. The *Generalized Argumentation Framework* (GAF) [38] extends traditional AFs with bipolarity (the possibility of expressing both attacks and supports between pairs of arguments) and weights on both the arguments and the attack/support relationships (denoting their strength). It provides a much more powerful model to carry out abstract argumentation, and is compatible with the most prominent extensions proposed in the literature.

Definition 10 ([38]). *A* Generalized Argumentation Framework *(**GAF**) is a tuple to begin* $F = \langle \mathcal{A}, \mathcal{S}(\mathcal{A}), w_\mathcal{A}, w_\mathcal{R} \rangle$, *where:*

- $\mathcal{A}$ *is a finite set of arguments,*
- $\mathcal{S}(\mathcal{A})$ *is a (possibly empty) system providing external information on the arguments in* $\mathcal{A}$,
- $w_\mathcal{A} \colon \mathcal{A} \times \mathcal{S}(\mathcal{A}) \mapsto [0, 1]$ *assigns a weight to each argument, to be considered as its intrinsic strength, also based on* $\mathcal{S}(\mathcal{A})$, *and*
- $w_\mathcal{R} \colon \mathcal{A} \times \mathcal{A} \mapsto [-1, 1]$ *assigns a weight to each pair of arguments.*

Quite intuitively, negative weights denote attacks (attacking an argument subtracts to its credibility) and positive weights denote supports (supporting an argument adds to its credibility). Weight 0 can be interpreted as the absence of any (attack or support) relationship (0-valued arcs are usually not drawn in the graph). Combinations of attacks and supports can also be considered:

1. Attacking the attacker of an argument amounts to defending (i.e., somehow supporting) that argument (known as *reinstatement*);
2. Attacking the supporter of an argument amounts to attacking that argument;
3. Supporting the attacker of an argument amounts to attacking that argument;
4. Supporting the supporter of an argument amounts to supporting that argument;

and is easily handled in GAFs using the mathematical sign rule:

followed by	Support	Attack
Support	Support	Attack
Attack	Attack	Support

$\equiv$

$\times$	$+$	$-$
$+$	$+$	$-$
$-$	$-$	$+$

(see Figure 1 for a graphical representation). This rule also holds for sequences of mixed attacks and supports: a sequence including an even number of attacks amounts to a defense (and indeed a product involving an even number of negative factors yields a positive result); vice versa, a sequence including an odd number of attacks still amounts to an attack (and indeed a product involving an odd number of negative factors yields a negative result). 0-weight links in a sequence would bring the product at 0, meaning that the initial and final arguments do not affect each other along that sequence.

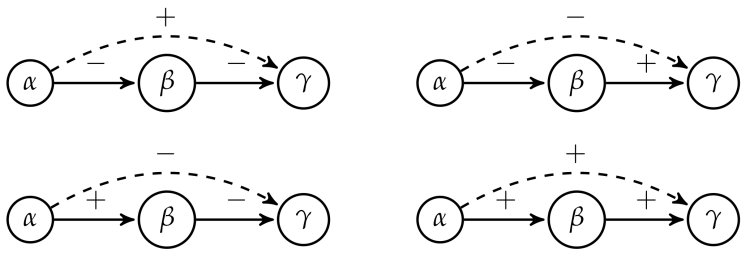

Figure 1. Sign rule for attacks and supports.

Having a bounded weight range, with fixed minimum and maximum values, can help intuition. -1 means that the attacking argument 'fully' defeats the attacked one; $+1$ means that the supporting argument 'fully' supports the supported one. The $]0, 1]$ range for attacks and supports is also very intuitive.

In real-world situations, often the context of the arguments affects their reliability. Contextual factors include the community in which the argumentation takes place, and the topic about which the claims are made. To take this into account, the T-GAFs specialization of the GAF model introduces community and topics as components in $\mathcal{S}(\mathcal{A})$:

$\mathcal{U}$ the finite set of members of the community, possibly including the entities who put forward the arguments, and

$\mathcal{T}$ a finite set of topics that may be involved in an argumentation (including a dummy topic $\top$ used to express the general authority and trust of a user, independent of specific topics).

In turn, these items allow to define some components to be used in $w_{\mathcal{A}}$, such as:

1. the subjective *confidence* that the members of the community (including the entity which posits the argument) have in an argument:
   $w_c : \mathcal{U} \times \mathcal{A} \mapsto [0, 1]$ where 1 means certainty, according to the classical probabilistic interpretation.

2. the recognized degree of *authority* on the topic of the argument of the entity putting it forward:
   $w_a : \mathcal{U} \times \mathcal{T} \mapsto [0, 1]$ where 1 means maximum authority of the user in the topic, and 0 absolutely no authority.
3. the *trust* that the community members have in the entity putting forward an argument, relative to the topic of the argument (indeed, not just the quality of evidence, but also the credibility of the entity positing it is important):
   $w_t : \mathcal{U} \times \mathcal{T} \mapsto [-1, 1]$ where -1 means total distrust, 0 means no opinion, and 1 means full trust.

(2) and (1) express the degree of expertise of an entity about a topic and its degree of confidence about a specific claim, respectively, and (3) expresses the degree of confidence by which a user's opinions about a topic are taken into consideration by other users. The assessment $w_{\mathcal{A}}$ of the 'intrinsic' reliability of an argument in the GAF may include these components in an overall formula once specific definitions for functions w_c, w_a and w_t (for the various topics) are given.

3.6. Induction

The term *induction* refers to the inference of general rules or theories starting from specific instances. *Observations* are descriptions of objects or situations as 'perceived' from the world. *Examples* are labels assigned to observations to explicitly specify what are the concepts of interest (to be learned) in the observations. Examples can be positive (representing instances of the concepts) or negative (representing instances that do not belong to a concept). Based on these definitions, *Inductive Learning* aims, given a set of examples concerning some concept, at extracting a model (i.e., a characterization) of that concept, on whose ground trying to foresee if new observations that are available correspond to the concept or not. Specifically, examples are used in supervised learning, while unsupervised learning is based on observations only.

Inductive Logic Programming (*ILP*) is a branch of Machine Learning concerned with the induction of models from positive and negative examples, exploiting the clausal fragment of FOL as a representation language for both the background knowledge and the induced theories. This setting has two main advantages: first, the induced models (and the information handled in general) have an intuitive meaning for humans, which makes them highly suitable for exploitation in application domains and systems for which the human validation of the machine processing is required; second, hard real-world domains requiring the representation of relations among objects can be faced.

In ILP, the observations are expressed as sets of facts, and do not consist of a fixed number of attributes, but may be longer or shorter, depending on what information they need to express. Some approaches to concept learning require that each example is explicitly associated with all the facts that are relevant to their underlying concept (*direct relevance* [39]). Others do not provide such explicit connection, and let the inductive system extract what it considers as relevant from an overall set of facts (*indirect relevance* [40]).

Definition 11 (Inductive Learning paradigm [13])**.** *A* theory *T is a set of hypotheses. A* hypothesis *H is a set of program clauses with the same head, i.e., defining the same concept. An* example *E is a ground (Horn) clause. It is called* positive *for a hypothesis H if its head has the same predicate and sign as H; it is called* negative *for H if its head has the same predicate as H but opposite sign.*

Given:

- *A set of examples $E = E^+ \cup E^-$,*
  where E^+ is a set of positive examples, and E^- a set of negative ones;
- *A (possibly empty)* background knowledge *(or* BK*) B.*

***Find** a theory (logic program, model) T s.t.*

- $T \cup B \vdash E^+$ *(*completeness *or* sufficiency*)*

- $T \cup B \cup E^- \not\vdash \square$ *((*prior *or* strong*)* consistency*)*

and moreover the following properties are fulfilled:

- $B \not\vdash E^+$ *((*prior*)* necessity*)*
- $B \cup E^- \not\vdash \square$ *(*prior consistency*)*
- $T \cup B \not\vdash \square$ *(*weak consistency*)*

As in deduction, the fundamental operation is unification [12] , allowing to find a common instance of two clauses (if any), in induction its dual, *generalization* [13] , allows to find (if any) a clause of which two given clauses are both instances. Just such as when one is interested in the most general unifier, not to skip any step in the deductive process, in the inductive process one looks for the *least general* generalization (*lgg*). It depends on the generality order adopted on clauses. The *lgg* of any two clauses under θ-subsumption, if any, is unique. Still, some efficiency and intuition issues led to the definition of a specialization of it, θ_{OI}-subsumption, based on the Object Identity assumption (see Section 2.2).

Definition 12 ([14]). *Given a theory T and an example E:*

- *T makes a* commission error *iff* $\exists H \in T, \exists C \in H$*: C is inconsistent wrt E*
- *T makes an* omission error *iff* $\exists H \in T$*: H is incomplete wrt E.*

An incomplete or inconsistent theory is incorrect*.*

An inconsistent (or *too strong*) theory is too general and needs to be specialized; an incomplete (or *too weak*) theory is too specific and needs to be generalized.

Some inductive systems work in a *batch* way: they start from an empty theory and stop the inductive process when the current set of hypotheses is able to explain all the available examples. When new evidence contradicts the learned theory, the whole process must be restarted from scratch, taking no advantage of the previously learned hypotheses. Other systems can revise and refine a theory in an *incremental* way: they try to change the previously generated hypotheses in such a way that the changed hypotheses explain both the old and the new examples. The former approach generally builds definitions top-down (i.e., from more general to more specific ones); it yields more compact and elegant theories, but is computationally expensive. The latter can just revise an existing hypothesis on the ground of new evidence, thus working bottom-up, but the resulting theory is not as elegant as those produced by batch systems. A cooperation between the two might run batch learning from time to time to obtain a better structured theory after many incremental revisions.

In the incremental approach, when the theory is incorrect it can/should be revised, searching for either a specialization (*downward refinement*) or a generalization (*upward refinement*) of the incorrect part of the theory. The refinement should be *minimal* [41]. Commission errors can be solved by exploiting properly a downward refinement operator, while, dually, upward refinement operators can cope with omission errors:

Definition 13 (Refinement operators [13]). *Let* $(S, \leq)$ *be a quasi ordered set.*
A downward refinement operator ρ *is a mapping s.t.* $\forall C \in S : \rho(C) \subseteq \{D \in S \mid D \leq C\}$*, i.e., it computes a subset of all specializations of C. An* upward refinement operator δ *is a mapping s.t.* $\forall C \in S : \delta(C) \subseteq \{D \in S \mid C \leq D\}$*, i.e., it computes a subset of all generalizations of C.*

3.7. Ontological

Born as a philosophical discipline that deals with the nature and structure of reality, ontologies were transposed to the computational domain and given more technical definitions depending on different perspectives. Here, we will adopt the perspective of "a formal, explicit specification of a shared conceptualization" [42], where a conceptualization is "an abstract, simplified view of the world that we wish to represent for some purpose" [9] that underlies all developments in Computer Science, and especially those of Knowledge Bases. Thus, an ontology describes the kinds of entities that are of interest in a domain,

their properties and relationships. Ontologies are sometimes intended as including the specific instances of those classes and relationships. In such a case, the definitional part is called *T-box* (for 'terminological'), the factual part about the instances is called *A-box* (for 'assertional'), and their union is called a *Knowledge Base* (KB).

Ontologies are important because they provide meaning and context to the symbols used in the KB . Typical ontology-based reasoning tasks of interest are satisfiability (checking if the described world may exist), instance checking (checking whether an instance belongs to a certain concept), concept satisfiability (checking if a concept may exist in the described world), subsumption (checking if a concept is a subclass of another concept), equivalence (checking if two classes are the same), retrieval (of the set of instances that belong to a certain concept), extraction of super-/sub-classes, relationships and properties of a given concept. Particularly relevant is the relationship of generalization/specialization, on which inheritance can be applied.

The research on ontologies evolved separately from LP, and relied on the Description Logics [43] fragment of FOL. Different description logics can be defined, depending on the available operators. Adding more and different operators extends the expressive power of a DL, but may lead to indecidability. Unfortunately, DLs are only partially overlapping to LP, and the non-overlapping parts are incompatible, mainly due to the different fundamental assumption they make on unknown information (Open World Assumption in DLs vs. Closed World Assumption in LP). Ontologies can be translated to default logic [44], one of the most famous formalisms for non-monotonic reasoning.

3.8. Similarity

Similarity computation between FOL descriptions is complex due to *indeterminacy* (the possibility of mapping various portions of one description in many ways onto another description). For this reason, very few works in the literature tackled this problem. Here, we will describe the approach proposed in [45] for *linked* Horn clauses, that was successfully used in many tasks. It considers a set of parameters and defines a similarity function based on them, plus a set of criteria to assess the similarity for different clause components. The parameters it uses for comparing two objects i' and i'' are widely accepted in the literature [46]:

n, the number of features owned by i' but not by i'' (*residual* of i' wrt i'');

l, the number of features owned both by i' and by i'';

m, the number of features owned by i'' but not by i' (*residual* of i'' wrt i').

Indeed, other classical and state-of-the-art distance measures are based on these parameters: e.g., the one developed by Tverski [47], by Dice ($S = 2l/(n + 2l + m)$) or Jaccard's index ($J(A, B) = l/(n + l + m)$) and distance ($J_\delta(A, B) = 1 - J(A, B) = (n + m)/(n + l + m)$). Instead, the similarity function is novel, to overcome the shortcomings of these functions:

$$\mathrm{sf}(\alpha, i', i'') = \mathrm{sf}(\alpha, n, l, m) = \alpha \frac{l+1}{l+n+2} + (1-\alpha)\frac{l+1}{l+m+2} \in]0,1[\quad (2)$$

where larger values denote more similarity, and α ($0 \leq \alpha \leq 1$) allows one to weight the importance of i' and i'' in the similarity computation.

The similarity criteria deal with increasingly complex clause components: terms, atoms, groups of atoms, clauses. The similarity of more complex components is based on the similarity of simpler components.

In FOL formulæ, terms represent specific objects, while predicates express their properties and relationships. Accordingly, two levels of similarity can be defined for pairs of FOL descriptions: the *object* level, concerning similarities between the terms referred to in the descriptions, and the *structure* one, referring to how the nets of relationships in the descriptions overlap.

In the case of Horn clauses, since the head consists of a single literal (and hence can be uniquely matched), it can be used as a starting point for the comparison. The proposed approach considers as comparable only clauses having atoms of the same arity n in the head, and the comparison outcome will be interpreted as the degree of similarity between the two n-tuples of terms in the heads.

Similarity between two clauses is computed level-wise, based on the similarity of their constituents. While the details of the computation procedure are not relevant here, we describe the principles underlying these comparisons.

Terms Given two terms t' and t'', their *object similarity* $\mathrm{fs}_o(t', t'')$ is computed based on a combination of two partial similarities: a *characteristic similarity* $\mathrm{fs}_c(t', t'')$ based on the properties they own (i.e., the unary predicates of which they are arguments) and a *role similarity* $\mathrm{fs}_r(t', t'')$ based on the roles they play in relation to other terms (i.e., the positions they occupy among the arguments of n-ary predicates).

Atoms Given two atoms l' and l'' built on n-ary predicates, their *star similarity* $\mathrm{fs}_s(l', l'')$ is computed based on a combination of: (a) the predicates of the atoms that are directly connected to them via some shared argument, and (b) the similarities of the terms appearing as their arguments.

Atom Sequences Given two sequences of atoms P' and P'' built on n-ary predicates, their *path similarity* $\mathrm{fs}_p(P', P'')$ is computed based on the initial parts of the sequences that can be mapped onto each other, combined with the star similarities of the mapped atoms and with the object similarity of the terms they involve.

Clauses whose similarity is based on the amounts of common and different atoms and terms in the two clauses, considering the least general generalization to determine an overall atom sequence for the clauses (see below) .

At each level, parameters n, l and m for the objects under comparison can be extracted to apply the similarity function.

All levels except the first one are based on the structure of a clause, defined by the way in which atoms built on n-ary predicates relate the various terms in the clause. Intuitively, the star of an atom depicts 'in breadth' how it relates to the rest of the formula, while a path in a clause depicts 'in depth' a given portion of the relations it describes. Multisets are needed since a predicate can have multiple instantiations among the considered atoms.

Clauses are represented by their *associated graph*, in which atoms are the nodes and arcs connect two nodes when their atoms share at least one argument. The sequences are obtained by building the *associated graph* as a Directed Acyclic Graph (DAG), *stratified* (i.e., with the set of nodes partitioned) in *levels* (elements of the partition) as follows. The head is the only node at level 0 (first element of the partition). It represents the unique starting point for building the associated graph, which also gives a unique access point for traversing it according to precise directions represented by the directed edges. Then, each successive level includes new nodes (not present in previous levels) that have at least one term in common with nodes in the previous level. In particular, each node (atom) in the new level has an incoming edge from each node (atom) in the previous level, having some argument (term) in common with it.

As said, the overall similarity between two clauses is computed based on their *least general generalization*:

Definition 14 ([45]). *Given two clauses C' and C'' with heads l'_0 and l''_0 respectively, call $C = l_0$:- $l_1, \ldots, l_k$ their least general generalization, and consider the substitutions θ' and θ'' such that $l_0\theta' = l'_0$, $l_0\theta'' = l''_0$ and $\forall i = 1, \ldots, k : l_i\theta' = l'_i \in body(C')$ and $l_i\theta'' = l''_i \in body(C'')$, respectively. The formula for assessing the overall similarity between C' and C'', called* formulæ similitudo *and denoted* fs*, is the following:*

$$\mathrm{fs}(C', C'') = \mathrm{sf}(n, l, m) \cdot \mathrm{sf}(n_o, l_o, m_o) + C^c(\{\mathrm{fs}_s(l'_i, l''_i)\}_{i=1,\ldots,k})$$

where $n = |body(C')| - |body(C)|$, $l = |body(C)| = k$, $m = |body(C'')| - |body(C)|$; $n_o = |terms(C')| - |terms(C)|$, $l_o = |terms(C)|$, $m_o = |terms(C'')| - |terms(C))|$; *and* C^c *is a function combining the values of the star similarities (e.g., the average).*

3.9. Analogy

Analogy is the cognitive process of matching the characterizing features of two items (subjects, objects, situations, etc.). It allows one to reuse knowledge from a known item or domain (called the *base*) to an unknown one (called the *target*), without having to learn from scratch. After finding an analogy on some roles, the association can be extended to further missing features. Analogical reasoning is essential for producing new conclusions that can help in solving a problem [48]. While similarity is a syntactic task that looks for exactly the same features in two items, analogy maps 'roles', which has to do with semantics (i.e., the meaning). The mapping is bi-directional, while in metaphors it only holds in one direction. The analogy may depend on the context, goal or perspective, and may lose its original meaning and usefulness if they change. Furthermore, the outcome of reasoning by analogy might be inconsistent with previous knowledge. "If the inferences mandated by an analogy contradict a fundamental belief, especially one that has accrued many consequent implications , then resolving this contradiction might well involve the *shock and amazement* of transformational creativity" [49]. However, a set of analogical mappings can be used to identify a *recurring meta-pattern* (i.e., a common network of roles).

Analogical reasoning consists of 5 steps [48]:

1. *Retrieval* finds the best base domain that may help to solve the problem in the target domain;
2. *Mapping* looks for a mapping between base and target domains;
3. *Evaluation* provides criteria to evaluate candidate mappings;
4. *Abstraction* shifts the representation of both domains to their roles' schema, converging to the same analogical pattern;
5. *Re-representation* adapts one or more pieces of the representation to improve the matching.

e.g., for Evaluation [48]: *structural soundness* holds if the alignment and the projected inference are structurally consistent; *factual validity* is needed to check if the projected inference preserves truth (i.e., it does not violate any constraint), which is not ensured because analogy is not a deductive mechanism (but just a hypothesis); *relevance for current goal* holds if and only if the produced inference moves the knowledge towards the goal.

Several works on analogy used non-relational representations, but are out of scope for this paper, which focuses on symbolic relational representations. The main contributions adopting formal (Propositional or First-Order) Logic representations include the *Structure Mapping Engine* (SME) [50–52] and its offspring [53,54], *Analogical Constraint Mapping Engine* (ACME) [55], *Learning and Inference with Schemas and Analogies* (LISA) [56], *Discovery Of Relations by Analogy* (DORA) [57], *Heuristic-driven Theory Projection* (HDTP) [58]. Most of them require that the same feature names are used for analogous roles, thus having more to do with similarity than to analogy. Others can map objects and relations having different names in the two domains, but use a quite complex knowledge representation formalism (e.g., SME and HDTP use dedicated sections of knowledge to formalize types of objects, or functions). HDTP also allows that features having the same name play different roles in the two domains.

A setting specifically based on LP formalisms for analogy was provided in [59]. Let us start from the formal definition of a role. Informally, a role represents a kind of interaction. These interactions are associated to the properties and relationships expressed by the predicates in a description. The set of roles of a term expresses the allowed interactions of that term.

Definition 15 (Roles [59]). *A* role *is a predicate* p/n *or its* r*-th argument position, denoted* $p/n.r$*; from this we define:*

- *The set of roles of a literal* l *such that* $\text{predicate}(l) = p/n$ *as* $R(l) = \{p/n\} \cup \{p/n.i\}_{i=1,\ldots,n}$;

- *The set of roles of a clause C as* $R(C) = \cup_{l \in C} R(l)$.

Concerning a specific term t, we have:

- *Given a literal l such that* predicate$(l) = p/n$, *the* literal roles *of a* $t \in$ terms(l) *are defined as* $R(t, l) = \{p/n.i \mid t \text{ is the i-th argument of } l\}$;
- *Given a clause C, the* clause roles *of a* $t \in$ terms(C) *are defined as* $R(t, C) = \cup_{l \in C} R(t, l)$;
- *A* role *of a term t in C is any element of* $R(t, C)$.

So, any predicate p/n defines $n + 1$ roles: p/n and $p/n.i$ $(i = 1, \ldots, n)$.

Being a mapping of roles across two domains, an analogy can be formalized as a mapping of predicates and terms that play the same role in two descriptions:

Definition 16 (Analogy [59])**.** *Given a pair of descriptions* $\langle D', D'' \rangle$ *and the associated sets of roles* $R(D')$ *and* $R(D'')$, *an* analogy *is a one-to-one mapping* $f : R' \to R''$ *for* $R' \subseteq R(D')$ *and* $R'' \subseteq R(D'')$. *If such an f exists,* D' *and* D'' *are analogous.*

After finding a mapping, the source domain/description might contain knowledge that is not present in the target one, and vice versa. In such a case, the missing knowledge in each domain might be 'borrowed' from the other. This would allow one to better understand each domain and in some cases to accomplish tasks (e.g., making comparisons, solving problems, etc.) that would be impossible without that additional knowledge. This opportunity is formalized as follows:

Definition 17 (Analogical perspective [59])**.** *Given a pair of descriptions* $\langle D_B, D_T \rangle$, *representing respectively the base and target domains, and an analogy* $f : K_B \subseteq R(D_B) \to K_T \subseteq R(D_T)$ *between* D_B *and* D_T, *let* S_T *and* S_B *denote the knowledge that solves a given task in* D_T *and the analogous task in* D_B, *respectively. f* satisfies *an* analogical perspective $P = \langle (D_B, S_B), (D_T, S_T) \rangle$ *if*

$$S_B \subseteq (K_B \cup \mathrm{t}_f^{-1}(I_T)) \wedge S_T \subseteq (K_T \cup \mathrm{t}_f(I_B))$$

where: K_B *and* K_T *denote the knowledge aligned by f;* $I_B \subseteq R(D_B) \setminus K_B$ *and* $I_T \subseteq R(D_T) \setminus K_T$ *are the non-aligned but transposable knowledge from the base and target domain, respectively; and the function* $\mathrm{t}_f : R(D_B) \to R(D_T)$ *transposes knowledge from* D_B *to* D_T *based on f.*

A procedure that, applied to two descriptions, returns possible analogies between them without requiring any additional knowledge is called an *analogy operator*. Reference [59] defined an analogy operator named *Roles Mapper*, that includes all of the above elements. It is based on a syntactic logical approach that, given two clauses, looks for analogical mappings in the body. The head labels the situation that is described in the body and may be used to provide a preferred perspective for which the analogical mapping is sought. If the arity of the predicates in the heads is the same, the system is bound to establish an analogy between corresponding arguments. Using 0-ary predicates in the heads disables this feature.

4. Multistrategy Reasoning

There are many interconnections among the inference strategies proposed above. Most of them may help the others in accomplishing their tasks. Their referring to the same framework, Logic Programming, enables these interactions. In the following, we will first describe existing attempts to combine partial groups of inference strategies, selected so that they may be combined all together in the spirit of true MultiStrategy Learning. We will also hypothesize new possible combinations, and quickly introduce the GEAR system that implements our vision.

4.1. Relevant Approaches to Combine Different Strategies

We will now report some examples proposed in the literature of fruitful cooperation among different strategies.

4.1.1. Ontologies and Logic Programming

While, as said, the realms of Logic Programming and Description Logics, used to specify ontologies, are partly incompatible, attempts to merge them have been made in the literature. Here, we mention $\mathcal{DL} + log$ [60], as the most powerful decidable combination of Description Logics and disjunctive Datalog rules (i.e., Datalog rules whose head may consist of a disjunction of atoms). This language distinguishes 'Datalog predicates', coming from the LP perspective, from 'DL predicates', coming from the ontological perspective. These predicates can be mixed in a clause, but DL predicates cannot be negated. Two safety conditions are set:

- (Datalog safeness) every variable occurring in the rule must appear in at least one of the positive literals in the body;
- (Weak safeness) every variable of the rule head must appear in at least one of the Datalog positive literals in the body.

Compared to other approaches, weak safeness allows one to express conjunctive queries in $\mathcal{DL} + log$ through weakly-safe rules, thus increasing the expressive power and still guaranteeing decidability for many DLs.

4.1.2. Abduction and Deduction

Abduction has been naturally defined in [23,24] in tight integration with deduction to allow reaching conclusions or making prediction when the available information is insufficient. Thus, we will not delve further into this specific combination.

4.1.3. Similarity for Deduction

Reference [45] shows how the similarity assessment approach can be used for helping a subsumption procedure to converge quickly towards the correct associations, and how it can be used to weaken subsumption and obtain a *flexible matching* procedure that returns a degree of matching instead of a boolean decision.

4.1.4. Abstraction, Deduction, Similarity, Abduction and Argumentation for Induction

A very interesting case of the use of several strategies in support of induction is provided by the incremental ILP system InTheLEx [61,62].

Abstraction is carried out as a pre-processing step that removes useless information according to the framework in [21]. This is obtained by expressing abstraction operators as clauses, such that whenever the body is recognized in an observation, the involved facts are replaced by those in the head, which is suitably defined to hide the useless information. In fact, there are several investigations in the literature on how abstraction can be used to shift to a higher-level language when concept descriptions that can explain the examples cannot be found using the current language [40,63–66].

Deduction is used in a *saturation* step that makes explicit facts that are implicit in the available description of the observations and that may be useful to correctly grasp the concept that is being learnt. To do this, deduction exploits the rules in the KB. Whenever the body of a clause in the theory is recognized in an observation, the head of the clause is added to the observation itself.

Abduction is used to check if an unexplained example/observation can be explained by assuming additional unseen information that is not present in the observations. In such a case, the guessed information is added to the example description. This prevents the refinement operators from being applied, and the theory from being changed.

Similarity is used to guide the generalization operator, by taking the paths univoquely determined according to the technique proposed in [45], and using a greedy techinque

that adds the generalization of these paths by decreasing similarity, as long as they are compatible. Further generalizations can then be obtained through backtracking [67].

Recently, argumentation has been integrated to identify consistent portions of inconsistent observations and to choose the one to rely on, exploiting the same integrity constraints defined for abduction to identify attacks and supports (an example of a further integration of different strategies) [68].

4.1.5. Induction for Abduction, Abstraction and Deduction

Abstraction operators, integrity constraints for abduction (and argumentation), and rules for saturation can be inductively learned from observations, as shown in [69,70]. Combinations of facts that never occur generate integrity constraints for abduction (that can be used also for generating abstract argumentation frameworks [68]). Combinations that always occur generate abstraction operators. Concept definitions learned by an ILP system can be used to identify known concepts in observations when learning other concepts.

4.1.6. Argumentation and Induction for Analogy

Reference [59] defined the *Roles Argumentation-based Mapper* analogy operator, that leverages argumentation to overcome the constraint that using the same descriptors in the two domains means that they necessarily denote the same roles. All possible analogical mappings between descriptors are considered, and mappings are inconsistent if they map one feature in one domain onto many features in the other. These inconsistencies are expressed as attacks in an argumentation framework, and abstract argumentation strategies are used to select only consistent associations.

In the same paper, the use of an inductive (generalization) operator to obtain more general knowledge structures that can be mapped onto several domains is also proposed.

4.1.7. Abduction and Probabilistic Reasoning

While, from a logical standpoint, all consistent abductive explanations are equally good, in a probabilistic setting different explanations of a goal are associated to different possible worlds, and their validity depends on the validity of the rules, facts and integrity constraints used to obtain those worlds . PEALP takes into account all these items [28,71]:

Definition 18 ([28]). *A* Probabilistic Expressive Abductive Logic Program *(PEALP) consists of a 4-tuple* $\langle P, A, I, p \rangle$*, where* P, A, I *as in EALP and* $p : P \cup \text{ground}(A) \cup I \rightarrow [0, 1]$ *is a probability function.*

Definition 19 ([28]). *Given a goal* G *and a PEALP* $T = \langle P, A, I, p \rangle$*, a* (probabilistic) abductive explanation *of (or* possible world *for)* G *in* T *is a triple* $E = (L, \Delta, \mathcal{C})$*, where* A *is the set of facts in* P*,* Δ *is the abductive explanation (i.e., the set of ground literals abduced), and* $\mathcal{C}$ *is the set of instances of (probabilistic) integrity constraints in* I*, involved in an abductive proof of* G *in* T*.*

A world that violates a probabilistic integrity constraint is not impossible, just differently probable. Thus, all different (minimal) abductive explanations must be obtained to identify the most likely one, and whenever the abductive procedure has a choice, it must explore the worlds associated to all different options. Given a PEALP $T = \langle P, A, I, p \rangle$, the *most likely explanation* for a goal G among all possible consistent worlds associated to G in T, $\mathcal{W}_G$, can be selected:

$$\Delta \text{ s.t. } \overline{E} = (L, \Delta, \mathcal{C}) = \arg \max_{E \in \mathcal{W}_G} p(E)$$

4.1.8. Cues for Further Cooperations

Interesting directions for combining different inference strategies are still to be thoroughly investigated. Here, we will envisage and propose some.

Analogy is still an open research direction, and thus its combination with other strategies is largely to be explored yet. As observed in the ITL, it is strictly connected to abstraction and deduction, since these two strategies are needed to go from a specific domain to its abstract structure and then from the latter to a new specific domain. It also has strict connections to abduction, that can be used to guess information in the target domain that is not observed but is analogous to information available in the source domain.

Uncertain reasoning is perhaps the most combinable strategy, allowing to add flexibiity to all the others. Especially promising to investigate are ways for combining it with argumentation (to determine how reliable each consistent setting is and rank different settings) and to induction (to assign a degree of reliability to the learned knowledge).

4.2. The GEAR MultiStrategy Reasoning Engine

GEAR (acronym for 'General Engine for Automated Reasoning') is an inference engine written in Prolog language, aimed at implementing the vision of LP-based MultiStrategy Reasoning envisioned and proposed in this paper. The current prototype brings to cooperation most of the strategies described in Section 4.1. Knowledge bases handled by GEAR may include various kinds of knowledge items, including Facts, Rules, Integrity Constraints, Abstraction Operators, and Argumentative relationships. Uncertainty is handled using values in $[0, 1]$, inspired by mathematical probability theory . Still, GEAR may adopt the more intuitive handling of uncertainty as proposed for MYCIN. In the following, we will briefly describe the main features of the formalism it uses.

The main components of a KB are facts and rules. Facts are formalized as

$$\texttt{fact([}M\texttt{,}I\texttt{], }F\texttt{, }C\texttt{).}$$

while rules are formalized as

$$\texttt{rule([}M\texttt{,}I\texttt{], }H\texttt{, }B\texttt{, }P\texttt{, }C\texttt{).}$$

where I is the unique identifier of the fact or rule, and $C \in]0, 1]$ is the certainty value (1 meaning 'absolutely' true and 0 meaning 'absolutely' false). F is an atom, while H and B are the rule's head and body, respectively, and P is its priority (a number used to determine which rule should be executed first in case of conflicts).

B is a logistic expression built on the following operators:

`and([`C_1`,...,`C_n`])` representing the conjunction (AND) of the C_i's;

`or([`C_1`,...,`C_n`])` representing the disjunction (OR) of the C_i's;

`no(`C`)` representing a 'probabilistic' negation (NOT) of C;

`not_exists(`C`)` representing an 'existential' negation of C;

where the C's are atoms or nested operator applications, to express complex conditions. H is one of the following:

C an atom;

`and([`C_1`,...,`C_n`])` representing the conjunction (AND) of the atoms C_i;

`or([`C_1`,...,`C_n`])` representing the disjunction (OR) of the atoms C_i;

`no(`C`)` representing a 'probabilistic' negation (NOT) of the atom C.

Abducibles are formalized as

$$\texttt{abducible (}P\texttt{/}N\texttt{).}$$

where P is the predicate name and N is its arity, while integrity constraints for abduction and argumentation are formalized as

$$\texttt{ic(}I\texttt{, }O\texttt{, }C\texttt{).}$$

where I is the unique identifier of the constraint, C is its certainty value, and O is one of the following:

`nand([`l_1`,...,`l_n`])` at least one among literals l_1,...,l_n must be false (the classical ICs considered in ALP);

`xor([`l_1`,...,`l_n`])` exactly one among literals l_1,...,l_n must be true;

`or([`l_1`,...,`l_n`])` at least one among literals l_1,...,l_n must be true;

`if([`l'_1`,...,`l'_n`],[`l''_1`,...,`l''_m`])` if all literals l'_1,...,l'_n are true, then all literals l''_1,...,l''_m must also be true (*modus ponens*); alternatively, if all literals l''_1,...,l''_m are false, then all literals l'_1,...,l'_n must also be false (*modus tollens*);

`iff([`l'_1`,...,`l'_n`],[`l''_1`,...,`l''_m`])` either all literals l'_1,...,l'_n and l''_1,...,l''_m are true, or all literals l'_1,...,l'_n and l''_1,...,l''_m are false;

`and([`l_1`,...,`l_n`])` all literals l_1,...,l_n must be true;

`nor([`l_1`,...,`l_n`])` all literals l_1,...,l_n must be false.

Abstraction operators are formalized as

$$\texttt{abstraction([}M\texttt{,}I\texttt{], }A\texttt{, }G\texttt{).}$$

where I is the unique identifier of the operator, and A is the abstracted set of atoms that replaces the ground set of atoms G whenever it is found in an observation.

Identifiers of knowledge items are in either of the following forms:

I a general unique identifier for the item in the overall KB;

`[`M`,`I`]` I is the unique identifier of the item within knowledge module M.

Finally, argumentation works on the following predicate:

$$\texttt{arg(}I\texttt{, }S\texttt{).}$$
$$\texttt{arg\_rel(}I'\texttt{, }I''\texttt{, }S'\texttt{).}$$

where I, I' and I'' are fact identifiers, $S \in [0,1]$ is the strength of the argument and $S' \in [-1,+1]$ expresses the type (attack or support, based on the sign) and strength of the argumentative relationship.

Other predicates can be used to specify system settings (e.g., `gear_flag` allows to set flags that direct the system's behavior), information related to user interaction (e.g., `askable` specifies information that can be asked to the user if missing in the KB), calls to pre-defined procedures (e.g., `call` may call Prolog to carry out some computations), and others, but they are beyond the scope of this paper.

For deduction, GEAR may work both forward (applying all the rules in the KB in order to derive all possible consequences of the initial set of facts) or backward (starting from a goal and focusing only on the deductive steps that are relevant to prove that goal). For abstraction, it works only in forward mode. For abduction, it works only in backward mode. For induction, GEAR exploits the mentioned ILP system InTheLEx: it is a supervised incremental learning system based on Datalog under Object Identity as a representation language. It can distinguish background knowledge, that is immutable, from the portion of the theory to be learned and refined. The typical information flow in InTheLEx is as follows. Every incoming example immediately undergoes abstraction, that eliminates uninteresting details according to the available operators. Then, the example is (deductively or abductively) checked against the current theory and the background knowledge. If the theory is incorrect, it must be refined, and thus the example is (abductively or deductively) saturated and the generalization or specialization operator is started to refine the theory, possibly using the abductive or deductive derivation whenever needed.

5. Conclusions

The symbolic/logic approach to AI, based on the First-Order Logic (FOL) setting, can handle relational representations of the data and reproduce high-level, conscious, human reasoning mechanisms. This allows the AI systems based on this approach to explain their behavior and decisions in human-understandable terms, which is fundamental when using AI in critical tasks of the real world, in order to enforce trustworthiness and support accountability. Much research has been carried out in the last decades on the definition and implementation of frameworks and operators that may simulate different inference strategies used by humans (deduction, abduction, abstraction, induction, etc.). Still, they were investigated separately or at most in small combinations. This paper claimed the need for an overall approach that merges all the single strategies, that we named *MultiStrategy Reasoning*. It identified Logic Programming as the most suitable setting to support this view, selected the most relevant and promising approaches for the single strategies developed in this setting , and described several kinds of combinations that can be merged into an overall approach, which is being implemented in the GEAR inference engine. The current version of GEAR is being used in various projects for providing explainable decision support in complex real-world tasks.

A systematic literature review was carried out and, to the best of our knowledge, this is the first proposal and attempt to define and implement an overall MultiStrategy Reasoning solution. While the set of strategies currently combined is wide and relevant, further work may expand it, improve their combination, and/or obtain more effective and efficient implementations thereof.

Funding: This research received no external funding.

Acknowledgments: The author would like to thank all his collaborators, colleagues and students that worked with him on these topics since 1995, for the useful discussions and the alternative perspective they suggested on many topics.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MSR	MultiStrategy Reasoning
FOL	First-Order Logic
KB	Knowledge Base
LP	Logic Programming
DL	Description Logics

References

1. Minsky, M. Logical versus analogical or symbolic versus connection or neat versus scruffy. *AI Mag.* **1991**, *12*, 34–51.
2. Yalçin, O.G. Symbolic vs. Subsymbolic AI Paradigms for AI Explainability. 2021. Available online: https://towardsdatascience.com/symbolic-vs-subsymbolic-ai-paradigms-for-ai-explainability-6e3982c6948a (accessed on 6 December 2022).
3. Minsky, M.; Papert, S. *Perceptrons*, 2nd ed.; MIT Press: Cambridge, MA, USA, 1988.
4. Barredo Arrieta, A.; Díaz-Rodríguez, N.; Del Ser, J.; Bennetot, A.; Tabik, S.; Barbado, A.; Garcia, S.; Gil-Lopez, S.; Molina, D.; Benjamins, R.; et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion* **2020**, *58*, 82–115. [CrossRef]
5. Phillips, P.J.; Hahn, C.A.; Fontana, P.C.; Yates, A.N.; Greene, K.; Broniatowski, D.A.; Przybocki, M.A. *Four Principles of Explainable Artificial Intelligence*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2021. [CrossRef]
6. The Royal Society. Explainable AI. 28 November 2019. Available online: https://royalsociety.org/topics-policy/projects/explainable-ai (accessed on 6 December 2022).
7. IBM. Explainable AI (XAI). Available online: https://www.ibm.com/watson/explainable-ai (accessed on 6 December 2022).
8. Turek, D.M. Explainable Artificial Intelligence (XAI). Available online: https://www.darpa.mil/program/explainable-artificial-intelligence (accessed on 6 December 2022).
9. Genesereth, M.; Nilsson, N. *Logical Foundations of Artificial Intelligence*; Morgan Kaufmann: Burlington, MA, USA, 1987.

10. Yalçin, O.G. 5 Significant Reasons Why Explainable AI Is an Existential Need for Humanity. 2020. Available online: https://towardsdatascience.com/5-significant-reasons-why-explainable-ai-is-an-existential-need-for-humanity-abe57ced4541 (accessed on 6 December 2022).
11. Michalski, R. Inferential Theory of Learning. Developing Foundations for Multistrategy Learning. In *Machine Learning. A Multistrategy Approach*; Michalski, R., Tecuci, G., Eds.; Morgan Kaufmann: San Mateo, CA, USA, 1994; Volume IV, pp. 3–61.
12. Lloyd, J. *Foundations of Logic Programming*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 1987.
13. Nienhuys-Cheng, S.; de Wolf, R. *Foundations of Inductive Logic Programming*; Lecture Notes in Artificial Intelligence; Springer: Berlin/Heidelberg, Germany, 1997; Volume 1228.
14. Ferilli, S. A Framework for Incremental Synthesis of Logic Theories: An Application to Document Processing. Ph.D. Thesis, University of Bari, Bari, Italy, 2000.
15. Helft, N. Inductive Generalization: A Logical Framework. In Proceedings of the Progress in Machine Learning, Bled, Slovenia, May 1987; pp. 149–157.
16. Ceri, S.; Gottlob, G.; Tanca, L. *Logic Programming and Databases*; Springer: Berlin/Heidelberg, Germany, 1990.
17. Rouveirol, C. Extensions of Inversion of Resolution Applied to Theory Completion. In *Inductive Logic Programming*; Academic Press: Cambridge, MA, USA, 1992; pp. 64–90.
18. Semeraro, G.; Esposito, F.; Malerba, D.; Fanizzi, N.; Ferilli, S. A Logic Framework for the Incremental Inductive Synthesis of Datalog Theories. In Proceedings of the 7th International Workshop on Logic Program Synthesis and Transformation, Leuven, Belgium, 10–12 July 1998; Volume 1463, pp. 300–321.
19. Reiter, R. Equality and Domain Closure in First Order Databases. *J. ACM* **1980**, *27*, 235–249. [CrossRef]
20. Clark, K.L. Negation as Failure. In *Logic and Databases*; Gallaire, H., Minker, J., Eds.; Plenum Press: New York, NY, USA, 1978; pp. 293–322.
21. Zucker, J.D. Semantic Abstraction for Concept Representation and Learning. In *Proceedings of the 4th International Workshop on Multistrategy Learning*; Michalski, R.S., Saitta, L., Eds.; Morgan Kaufmann Publishers: Burlington, MA, USA, 1998; pp. 157–164.
22. Giunchiglia, F.; Walsh, T. A Theory of Abstraction. *Artif. Intell.* **1992**, *57*, 323–389. [CrossRef]
23. Kakas, A.; Kowalski, R.; Toni, F. Abductive Logic Programming. *J. Log. Comput.* **1993**, *2*. 718–770. [CrossRef]
24. Kakas, A.; Mancarella, P. Generalized Stable Models: A Semantics for Abduction. In Proceedings of the 9th European Conference on Artificial Intelligence, Stockholm Sweden, 1 January 1990; pp. 385–391.
25. Denecker, M.; Kakas, A. Abduction in Logic Programming. In *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski*; Lecture Notes in Computer Science, Volume 2407; Springer: Berlin/Heidelberg, Germany, 2002; pp. 402–437.
26. Denecker, M.; Schreye, D.D. SLDNFA: An abductive procedure for normal abductive programs. In Proceedings of the ICSLP, Banff, AB, Canada, 12–16 October 1992; pp. 700–868.
27. Kakas, A.; Mancarella, P. On the Relation of Truth Maintenance and Abduction. In Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence, Nagoya, Japan, 14–16 November 1990.
28. Ferilli, S. Extending expressivity and flexibility of abductive logic programming. *J. Intell. Inf. Syst.* **2018**, *3*, 647–672. [CrossRef]
29. Riguzzi, F. *Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning*; River Publishers: Gistrup, Denmark, 2018.
30. Sato, T. A Statistical Learning Method for Logic Programs with Distribution Semantics. In Proceedings of the Logic Programming, Twelfth International Conference on Logic Programming, Tokyo, Japan, 13–16 June 1995; pp. 715–729.
31. De Raedt, L.; Kimmig, A.; Toivonen, H. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), Hyderabad, India, 6–12 January 2007; Volume 7, pp. 2462–2467.
32. Sato, T.; Kameya, Y. PRISM: A language for symbolic-statistical modeling. In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI 1997), Nagoya, Japan, 23–29 August 1997; Volume 97, pp. 1330–1339.
33. Vennekens, J.; Verbaeten, S.; Bruynooghe, M. Logic Programs With Annotated Disjunctions. In Proceedings of the 20th International Conference on Logic Programming (ICLP 2004), Saint-Malo, France, 6–10 September 2004; pp. 431–445. [CrossRef]
34. Vennekens, J.; Denecker, M.; Bruynooghe, M. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory Pract. Log. Program.* **2009**, *9*, 245–308. [CrossRef]
35. Shortliffe, E.H.; Buchanan, B.G. A model of inexact reasoning in medicine. *Math. Biosci.* **1975**, *23*, 351–379. [CrossRef]
36. Zadeh, L. Fuzzy sets. *Inf. Control* **1965**, *8*, 338–353. [CrossRef]
37. Dung, P.M. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and N-Person Games. *Artif. Intell.* **1995**, *77*, 321–357. [CrossRef]
38. Ferilli, S. Introducing General Argumentation Frameworks and their Use. In Proceedings of the AIxIA 2020 (Reboot)—The 19th International Conference of the Italian Association for Artificial Intelligence, Milan, Italy, 1–3 December 2021; Volume 12414, pp. 136–153.
39. Michalski, R.S. A Theory and Methodology of Inductive Learning. In *Machine Learning: An Artificial Intelligence Approach*; Michalski, R.S., Carbonell, J.G., Mitchell, T.M., Eds.; Morgan Kaufmann: San Mateo, CA, USA, 1983; Volume I.

40. De Raedt, L. *Interactive Theory Revision—An Inductive Logic Programming Approach*; Academic Press: Cambridge, MA, USA, 1992.
41. Wrobel, S. On the proper definition of minimality in specialization and theory revision. In Proceedings of the Machine Learning, Amherst, MA, USA, 27–29 June 1993; Number 667, pp. 65–82.
42. Studer, R.; Benjamins, V.; Fensel, D. Knowledge engineering: Principles and methods. *Data Knowl. Eng.* **1998**, *25*, 161–197. [CrossRef]
43. *The Description Logic Handbook: Theory, Implementation and Applications*, 2nd ed.; Cambridge University Press: Cambridge, MA, USA, 2007. [CrossRef]
44. Sun, Y.; Sui, Y. Translating Ontologies to Default Logic. In Proceedings of the AIAI, Innsbruck, Austria, 14–16 February 2005.
45. Ferilli, S.; Basile, T.; Biba, M.; Di Mauro, N.; Esposito, F. A General Similarity Framework for Horn Clause Logic. *Fundam. Informaticae* **2009**, *90*, 43–66. [CrossRef]
46. Lin, D. An information-theoretic definition of similarity. In Proceedings of the 15th International Conference on Machine Learning, Madison, WI, USA, 24–27 July 1998, pp. 296–304.
47. Tversky, A. Features of Similarity. *Psychol. Rev.* **1977**, *84*, 327–352. [CrossRef]
48. Gentner, D. Analogy. *A Companion to Cognitive Science*; Wiley: Hoboken, NJ, USA, 1998; pp. 107–113.
49. O'Donoghue, D.; Keane, M.T. A Creative Analogy Machine: Results and Challenges. In Proceedings of the Third International Conference on Computational Creativity, Dublin, Ireland, 30 May–1 June 2012; pp. 17–24.
50. Gentner, D. Structure-mapping: A theoretical framework for analogy. *Cogn. Sci.* **1983**, *7*, 155–170. [CrossRef]
51. Falkenhainer, B.; Forbus, K.D.; Gentner, D. The Structure-Mapping Engine: Algorithm and Examples. *Artif. Intell.* **1989**, *41*, 1–63. [CrossRef]
52. Gentner, D.; Markman, A.B. Structure mapping in analogy and similarity. *Am. Psychol.* **1997**, *52*, 45–56. [CrossRef]
53. de los Angeles Chang, M.; Forbus, K.D. Using Quantitative Information to Improve Analogical Matching Between Sketches. In Proceedings of the 24th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), Toronto, ON, Canada, 22–26 July 2012.
54. Baydin, A.G.; de Mántaras, R.L.; Ontañón, S. Automated Generation of Cross-Domain Analogies via Evolutionary Computation. *arXiv* **2012**, arXiv:1204.2335.
55. Holyoak, K.J.; Thagard, P. Analogical mapping by constraint satisfaction. *Cogn. Sci.* **1989**, *13*, 295–355. [CrossRef]
56. Holyoak, K.J.; Hummel, J.E. Understanding analogy within a biological symbol system. In *The Analogical Mind*; Dedre Gentner, K.J.H., Konikov, B.N., Eds.; The MIT Press: Cambridge, MA, USA, 2001; pp. 161–195.
57. Doumas, L.A.A.; Hummel, J.E.; Sandhofer, C.M. A theory of the discovery and predication of relational concepts. *Psychol. Rev.* **2008**, *115*, 1–43. [CrossRef]
58. Schwering, A.; Krumnack, U.; Kühnberger, K.U.; Gust, H. Syntactic Principles of Heuristic-driven Theory Projection. *Cogn. Syst. Res.* **2009**, *10*, 251–269. [CrossRef]
59. Leuzzi, F.; Ferilli, S. A multi-strategy approach to structural analogy making. *J. Intell. Inf. Syst.* **2018**, *50*, 1–28. [CrossRef]
60. Rosati, R. DL+log: Tight integration of description logics and disjunctive datalog. In Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District, UK, 2–5 June 2006; pp. 68–78.
61. Esposito, F.; Semeraro, G.; Fanizzi, N.; Ferilli, S. Multistrategy Theory Revision: Induction and Abduction in INTHELEX. *Mach. Learn. J.* **2000**, *38*, 133–156. [CrossRef]
62. Esposito, F.; Fanizzi, N.; Ferilli, S.; Basile, T.M.; Di Mauro, N. Multistrategy Operators for Relational Learning and Their Cooperation. *Fundam. Informaticae* **2006**, *69*, 389–409.
63. Utgoff, P. Shift of Bias for Inductive Concept Learning. In *Machine Learning: An Artificial Intelligence Approach*; Michalski, R., Carbonell, J., Mitchell, T., Eds.; Morgan Kaufmann: Los Altos, CA, USA 1986; Volume II, pp. 107–148.
64. Giordana, A.; Saitta, L. Abstraction: A General Framework for Learning. In Proceedings of the Working Notes of the Workshop on Automated Generation of Approximations and Abstractions, Boston, MA, USA, July 1990; pp. 245–256.
65. Bournaud, I.; Courtine, M.; Zucker, J.D. Abstractions for Knowledge Organization of Relational Descriptions. In Proceedings of the Abstraction, Reformulation, and Approximation, 4th International Symposium, SARA 2000, Horseshoe Bay, TX, USA, 26–29 July 2000; Volume 1864, pp. 87–106.
66. Bournaud, I.; Courtine, M.; J.-D., Z. Propositionalization for Clustering Symbolic Relational Descriptions. In Proceedings of the Twelfth International Conference on Inductive Logic Programming (ILP 2002), Sydney, Australia, 9–11 July 2002.
67. Ferilli, S.; Basile, T.M.A.; Di Mauro, N.; Biba, M.; Esposito, F. Similarity-Guided Clause Generalization. In Proceedings of the AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, Rome, Italy, 10–13 September 2007; pp. 278–289.
68. Pazienza, A.; Ferilli, S. Exploring Abstract Argumentation-Based Approaches to Tackle Inconsistent Observations in Inductive Logic Programming. In Proceedings of the AI*IA 2018—Advances in Artificial Intelligence, Trento, Italy, 20–23 November 2018; pp. 279–292.
69. Ferilli, S.; Basile, T.M.A.; Di Mauro, N.; Esposito, F. On the LearnAbility of Abstraction Theories from Observations for Relational Learning. In Proceedings of the Machine Learning: ECML 2005, Porto, Portugal, 3–7 October 2005; pp. 120–132.

70. Esposito, F.; Ferilli, S.; Basile, T.; Di Mauro, N. Inference of abduction theories for handling incompleteness in first-order learning. *Knowl. Inf. Syst.* **2007**, *11*, 217–242. [CrossRef]
71. Azzolini, D.; Bellodi, E.; Ferilli, S.; Riguzzi, F.; Zese, R. Abduction with Probabilistic Logic Programming under the Distribution Semantics. *Int. J. Approx. Reason.* **2022**, *142*, 41–63. [CrossRef]