Combining Nonmonotonic Knowledge Bases with External Sources[⋆]

Thomas Eiter[1], Gerhard Brewka[2], Minh Dao-Tran[1], Michael Fink[1], Giovambattista Ianni[3], and Thomas Krennwallner[1]

[1] Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,dao,fink,tkren}@kr.tuwien.ac.at

[2] Universität Leipzig, Augustusplatz 10-11, 04109 Leipzig, Germany
brewka@informatik.uni-leipzig.de

[3] Dipartimento di Matematica, Universitá della Calabria, I-87036 Rende (CS), Italy
ianni@mat.unical.it

Abstract. The developments in information technology during the last decade have been rapidly changing the possibilities for data and knowledge access. To respect this, several declarative knowledge representation formalisms have been extended with the capability to access data and knowledge sources that are external to a knowledge base. This article reviews some of these formalisms that are centered around Answer Set Programming, viz. HEX-programs, modular logic programs, and multi-context systems, which were developed by the KBS group of the Vienna University of Technology in cooperation with external colleagues. These formalisms were designed with different principles and four different settings, and thus have different properties and features; however, as argued, they are not unrelated. Furthermore, they provide a basis for advanced knowledge-based information systems, which are targeted in ongoing research projects.

1 Introduction

The developments in information technology during the last decade have been rapidly changing the possibilities for data and knowledge access. The World Wide Web and the underlying Internet provide a backbone for the information systems of the 21st century, which will possess powerful reasoning capabilities that enable one to combine various pieces of information, possibly stored in heterogeneous formats and with different semantics, such that the wealth of information can be more profitably exploited. In that, information from plain sources and software packages with simple semantics (such as, e.g., from a route planner) will have to be mixed with semantically richer sources like expert knowledge bases, in a suitable manner, bridging the gap between different sources.

Driven by this need, extensions of declarative knowledge representation formalisms have been developed with the capability to access external data and knowledge sources.

[⋆] This work has been supported by the Austrian Science Fund (FWF) projects P20840 & P20841, the EC ICT Integrated Project Ontorule (FP7 231875), and the Vienna Science and Technology Fund (WWTF) project ICT08-020.

Often, this is realized via an interface in the style of an API; examples of such rule based formalisms are Prolog engines, or extensions of Answer Set Programming (ASP), which is based on nonmonotonic logic programs. A particular application area where extensions of nonmonotonic formalisms received a lot of attention recently is the Semantic Web, cf. [35,41,1], and especially combinations of rules with external ontologies; see [14,10] for overviews and discussions.

However, such extensions are non-trivial, especially if the flow of information between a knowledge base and the external sources is bidirectional. That is, the external source influences the reasoning of the knowledge base, which in turn influences the behavior of the external source. To define suitable semantics for such scenarios, in the presence of heterogeneity and distribution, is a challenging problem.

At the Knowledge Based Systems (KBS) Group of the Vienna University of Technology, people have been working on this problem in several past and ongoing projects, with a focus on combining nonmonotonic knowledge bases with external sources, in cooperation with other researchers. The aim of this article is to give a short survey of some of the formalisms that have been developed, and to provide (for the first time) a more systematic view of these approaches, according to some characteristic features which, on the one hand derive from the underlying setting and on the other hand also determine the properties of the formalisms. Comparison to related work will be largely omitted here, and we refer to the original papers for this.

Historic Background. The work at KBS on access to external sources in the last years has precursors dating back more than a decade ago, in different areas: the action language for the IMPACT agent platform [39] in the area of agents, and logic programs with generalized quantifiers [13], in the area of nonmonotonic logic programming.

The IMPACT agent language [20,39] is a rule-based language for specifying the behavior of a single agent, in terms of actions she may take, depending on the agent state and input perceived from the environment and other agents via a message box. As the agent program sits on top of internal data structures, access to such data structures via *code calls* (available through APIs) in special *code call atoms* had been devised. As these code calls are in fact logically independent of the physical realization, they can be equally viewed as access to external data sources (as done in some system demos). A suite of semantics has been defined for agent programs, including nonmonotonic ones like minimal model and stable semantics.

Nonmonotonic logic programs with generalized quantifiers (GQLPs) were proposed in [13] to increase the expressiveness of logic programs under answer set semantics, by incorporating *Generalized Quantifiers (GQs)* akin to Lindström quantifiers in first-order logic (such as majority quantifiers; see [40]); similar extensions had been conceived for database query languages like SQL, to model aggregate functions, or to incorporate transitive closure. Special *GQ atoms* allowed to evaluate GQs (which, semantically boils down to decide whether a particular structure, determined by input predicates, belongs to a class of first-order structures). Viewing logic programs as GQs, [13] developed an approach to modular logic programming in which a program module can access other modules through an interface, which returns inferences of a module depending on input provided by the calling module.

Table 1. Classification of formalisms

reduct / world view	GL-style	FLP
local model	GQLPs	HEX
globale state	MCS	MLPs

However, the formalisms in [20,13] have some limitations and shortcomings. Both suffer from groundedness issues in the semantics, in that atoms might be true in "models" without "founded" support in terms of rules that derive these atoms (a ubiquitous problem in knowledge representation and reasoning, most prominently discussed in Autoepistemic Logic; see [31]). Furthermore, in IMPACT, the focus was on efficient executability over (heterogenous) data structures, which was realized with rule unfolding and pre-compilation; only a very rudimentary (monotonic) fragment of the language was implemented. GQLPs were geared towards accessing sources with inherent logical properties of admissible classes of structures, and modular logic programs on top did not allow for recursion in module calls; furthermore, no implementation was available.[1]

Recent Work. Motivated by the growing desire for extensions of ASP to access external sources, which especially arose in the Semantic Web area, HEX-programs were proposed in [17] as a basic formalism for this purpose, abstracting from the more special description logic (dl-)programs [19] that combine ASP and OWL ontologies. To overcome the problems of modular logic programming via GQLPs, a refined approach has been recently presented in [7] which redefines the semantics of modular logic programs and, noticeably, allows for arbitrary (mutual) recursion between modules. Orthogonal to these formalisms are multi-context systems [2], which were motivated from a different angle, namely reasoning with contexts. Here, beliefs between several contexts, which can be seen as agents with different views of a scenario, have to be exchanged; naturally, this amounts to knowledge bases with external knowledge access, where nonmonotonicity is a desired feature.

As detailed in later sections, modular logic programs (MLPs) can be viewed as a special setting for HEX-programs where external sources are nonmonotonic logic programs themselves. In contrast, multi-context systems (MCSs) can be viewed as a generalization of HEX-programs, in which information exchange is moved to the meta-level above the knowledge bases that instantiate a generic logic. However, this view is superficial and neglects important aspects that make the formalisms rather different.

From a principled view, the most important are the following two orthogonal aspects:

environment view. The definition of the semantics takes either an individual or a societal view; in the former, even though there is a collection of programs (or knowledge bases) $KB_1, \ldots, KB_n$, the semantics is merely defined in terms of *local models* for each individual knowledge base KB_i; the semantics of the collection implicitly emerges from the local models. In contrast, in the societal view,

[1] The first systems supporting answer set semantics became available at that time; Gerald Pfeifer, one of the chief developers of the DLV system, deemed GQLPs in 1997 as very interesting and put a respective task on his growing todo list (it is still there).

the collection has a *global state*, which consists of a collection of local models, one for each KB_i, that is explicitly accessible. The latter allows, e.g., to define preference over global states, and to single out most preferred ones. Protagonists of the local-model semantics are GQLPs and HEX-programs, while MLPs and MCSs have global-state semantics. Loosely speaking, in game-theoretic terms the former semantics are akin to Nash equilibria, while the latter strive for *Pareto-optimality*.

program reduct. All formalisms involve, in the tradition of answer set semantics, a notion of reduct which alters the rules of a program. The classical definition of answer set semantics [24] uses the *Gelfond-Lifschitz (GL) reduct* [23], which given an interpretation roughly removes grounded rules whose negative body part is false in the interpretation, and strips off negative literals from the remaining rules. Later, the *Faber-Leone-Pfeifer (FLP) reduct* [21] was presented which simply removes all grounded rules with a false body. While the two reducts are equivalent for ordinary logic programs, they behave differently for language extensions; an attractive feature of the FLP reduct is that it retains minimality of models, which helps to ensure groundedness of the semantics. Of the formalisms considered here, GQLPs and MCSs use a GL-style reduct, while HEX-programs and MLPs use the FLP-reduct.

In summary, this leads to a systematic classification of approaches shown in Table 1. Each of the possible combinations, which comes with different features and properties, is in fact populated by a formalism from above. Other formalisms we developed also fit into this classification; e.g., dl-programs [19] are local-model/GL-style reduct.

The different combinations are not unrelated, and in some cases, one type of combination might coincide with another one or be reducible to it. For example, in special cases, the choice of the reduct does not play a role (this holds, e.g., for the premier fragment of dl-programs, cf. [17]). Furthermore, as we show in Section 5, under a natural condition MCSs can be encoded into HEX-programs. Finally, the classification also shows ways for possible variations of existing formalisms (e.g., MCS).

Roadmap. The rest of this article is structured as follows. In the next section, we recall the answer set semantics of nonmonotonic logic programs, where we provide both the original definition [24] and the equivalent one in terms of the FLP-reduct [21]. In the Sections 3–5, we then consider HEX-programs, modular logic programs, and multi-context systems and discuss their relationship. After that, we present in Section 6 ongoing work and projects, together with issues for research. The final Section 7 gives a short summary and conclusions.

2 Preliminaries

In this section, we recall the answer set semantics of logic programs (over classical literals) [24], which extends the stable model semantics [23] with *classical* (or, more appropriately, *strong*) *negation*. For more background, see [24,22,15].

Syntax. Ordinary logic programs are built over a first-order vocabulary Φ with nonempty finite sets $\mathcal{P}$, $\mathcal{C}$, $\mathcal{F}$ of predicate, constant, and function symbols (of arity $n \geq 1$), and a set $\mathcal{X}$ of variables. *Terms* are inductively built as usual from $\mathcal{C}$ and $\mathcal{X}$ using function symbols from $\mathcal{F}$. *Atoms* are expressions of the form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{P}$ has arity $n \geq 0$ and $t_1, \ldots, t_n$ are terms. A *classical literal* (simply *literal*) l is an atom α or a negated atom $\neg\alpha$; its *complement* is $\neg\alpha$ (resp., α). A *negation-as-failure literal* (or simply *NAF-literal*) is a literal l or a default-negated literal $not\, l$.

A *(disjunctive) rule* r is of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_m, \text{not}\, \beta_{m+1}, \ldots, \text{not}\, \beta_n \tag{1}$$

where $k + n > 0$ and all α_i and β_j are literals. The disjunction $\alpha_1 \vee \cdots \vee \alpha_k$ is the *head* of r, and the conjunction $\beta_1, \ldots, \beta_m, \text{not}\, \beta_{m+1}, \ldots, \text{not}\, \beta_n$ is the *body* of r, where $\beta_1, \ldots, \beta_m$ (resp., not $\beta_{m+1}, \ldots,$ not β_n) is the *positive* (resp., *negative*) *body* of r. We use the notation $H(r) = \{\alpha_1, \ldots, \alpha_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \ldots, \beta_m\}$ and $B^-(r) = \{\beta_{m+1}, \ldots, \beta_n\}$.

If $B(r) = \emptyset$, then r is a *(disjunctive) fact*; we also omit "$\leftarrow$" in this case. If $H(r) = \emptyset$, then r is a *constraint*. If $k = 1$, then r is called *normal*, and if $m = n$, then r *is positive* (or not-*free*).

A *(disjunctive) program* is a finite set of rules. A program P is *normal* (resp., *positive*), if each rule $r \in P$ is *normal* (resp., *positive*).

While we have defined here programs with function symbols, traditional Answer Set Programming does not consider function symbols, as they lead to undecidability; however, more recently, decidable fragments have received attention, cf. [9]. Furthermore, Φ is often implicit from the rules of program P, i.e., $\Phi = \Phi_P$; if no constant appears in P, an arbitrary constant symbol is added to $\mathcal{C}$.

Semantics. The answer set semantics is defined in terms of consistent sets of classical literals. Positive programs are assigned the minimal consistent sets of classical ground literals that satisfy all rules; the semantics of arbitrary programs is defined by a reduction to positive programs.

As usual, a term, atom etc. is *ground*, if no variable occurs in it. Let HU_P be the *Herbrand universe* of a program P, which consists of all ground terms over Φ_P. The *Herbrand base* of P, denoted HB_P, is the set of all ground (classical) literals with predicate symbols from $\mathcal{P}$ and terms from HU_P.

An *interpretation* I relative to P is a consistent subset of HB_P. Satisfaction of ground literals, rules, and programs relative to I is as follows. I is a model of

- a ground literal α ($I \models \alpha$) iff $\alpha \in I$;
- a ground rule r ($I \models r$) iff $I \models H(r)$ whenever $I \models B(r)$, where (i) $I \models H(r)$ iff there is some $\alpha_i \in H(r)$ such that $I \models \alpha_i$, and (ii) $I \models B(r)$ iff $I \models \beta_j$ for all $\beta_j \in B^+(r)$ and $I \not\models \beta_j$ for all $\beta_j \in B^-(r)$.
- a set of ground rules R ($I \models R$) iff $I \models r$ for all $r \in R$.

Models of nonground rules r and programs P, are defined with respect to their groundings $grnd(r)$ and $grnd(P) = \bigcup_{r \in P} grnd(r)$, where $grnd(r)$ consists of all ground instances of r. A program P is (classically) *satisfiable*, if it has some model.

Then, for a positive program P, an *answer set* of P is any interpretation I such that $I \models P$ and $J \not\models P$ for every $J \subset I$, i.e., I is a minimal model of P under set inclusion.

Definition 1 (Gelfond-Lifschitz reduct). *The* Gelfond-Lifschitz reduct, *of a program P relative to an interpretation $I \subseteq HB_P$, denoted P^I, is the ground positive program that results from $grnd(P)$ by*

(i) deleting every rule r such that $B^-(r) \cap I \neq \emptyset$, and
(ii) deleting the negative body from every remaining rule.

An *answer set* of a (disjunctive) program P is any interpretation $I \subseteq HB_P$ such that I is an answer set of P^I. The set of all answer sets of a program P is denoted by $ans_{GL}(P)$.

Example 1. Consider the normal logic program P, consisting of the following rules, where g is an atom:

$$g \leftarrow \text{not } \neg g; \qquad \neg g \leftarrow \text{not } g. \tag{2}$$

Then, the answer sets of P are given by $M_1 = \{g\}$ and $M_2 = \{\neg g\}$. Informally, the rules allow to choose between g and $\neg g$; the single disjunctive fact $g \vee \neg g$ yields the same result.

Example 2. The following rules select from a set (stored in a predicate p) one element:

$$sel(X) \leftarrow p(X), \text{not } \neg sel(X).$$
$$\neg sel(X) \vee \neg sel(Y) \leftarrow p(X), p(Y), X \neq Y.$$

Informally, the first rule says that an element is picked by default, and the second that from two elements, at least one is not picked (here "$\neq$" is a built-in predicate that can easily be defined). Adding facts $F = \{p(c_i) \mid 1 \leq i \leq n\}$, the resulting program P has the answer sets $M_i = F \cup \{\neg sel(c_j) \mid 1 \leq j \neq i \leq n\} \cup \{sel(c_i)\}$, $i = 1, \ldots, n$.

We note that strong negation does not increase expressivity and can be easily compiled away, by viewing $\neg p$ as a fresh predicate symbol and adding the constraint $\leftarrow p(X_1, \ldots, X_n), \neg p(X_1, \ldots, X_n)$; thus, in ASP formalisms (e.g., in HEX-programs) strong negation is often omitted for simplicity.

Answer Sets using the FLP-reduct. Answer sets can be alternatively defined in many ways, cf. [29]. For our concerns, the following is of particular interest.

Definition 2 (Faber-Leone-Pfeifer reduct). *The* Faber-Leone-Pfeifer (FLP) reduct *of a program P relative to an interpretation $I \subseteq HB_P$, denoted fP^I, is the ground program consisting of rules $r \in grnd(P)$ such that $I \models B(r)$.*

An *FLP answer set* of a disjunctive program P is an interpretation $I \subseteq HB_P$ such that $I \models fP^I$ and no $J \subset I$ exists such that $J \models fP^I$. The set of all FLP answer sets of a program P is denoted by $ans_{FLP}(P)$. Thus, FLP answer sets differ from the usual ones only by the use of fP^I instead of P^I. Faber *et al.* [21] show that this is immaterial.

Proposition 1 ([21]). *For every (disjunctive) program P, $ans_{GL}(P) = ans_{FLP}(P)$.*

This property does not generalize to extensions of logic programs in which the building blocks α_i and β_j in a rule (1) may be other constructs than literals. This is the case e.g. for aggregate atoms [21], or for GQ atoms in [13] and external atoms in [17].

3 HEX-Programs

HEX-programs are a basic formalism featuring *a) external atoms* for accessing outer information in logic programs, and *b)* constructs for performing higher-order reasoning [17,18]. HEX-programs take inspiration and generalize their ancestors, such as the action language of the IMPACT agent platform [20,39], dl-programs [19,16] and the DLV-EX formalism [6,5]. Higher-order constructs enable a form of reasoning at the terminological level, overcoming some limitations of traditional logic programming under answer set semantics in this respect. This latter issue falls outside the scope of this paper; thus, among the two main extensions characterizing HEX-programs, we will focus next on external atoms.

3.1 Motivation and Outline

The conception of HEX-programs stems from some of its closest ancestors, that is dl-programs [19,16], and the DLV-EX extension to the DLV system [6,5].

Focusing on interoperability with description logic bases, dl-programs make use of *dl-atoms* to deal with this single species of external knowledge. Such atoms enable dl-programs to query an external source of knowledge, expressed in a description logic of choice, and allow a bidirectional flow of information about concept membership and role assertions to and from external sources. Notably, dl-programs assume a known and finite domain of individual constants. In general, invention of unknown values coming from external sources is both an important theoretical issue, and a desirable practical feature.

On the other hand, DLV-EX focused on the possibility to introduce general purpose external predicates, explicitly allowing to bring new constants from the outer realms into play. Nonetheless, this framework has a flow of information based on constants and values, without relations and higher order data as possible in HEX.

HEX-programs combine benefits of both frameworks within the notion of external atoms: there can be many sorts of external atoms, each of which is connected to a different kind of external knowledge and/or computation; also, it is possible to have relational information flow from and to the logic program at hand.

Informally, one can exploit external predicates through *external atoms* such as the RDF atom $\&rdf[urls, graphs](X, Y, Z)$. Here, $graphs$ and $urls$ constitute the input list, while X, Y and Z refer to the attributes of a ternary relation which can be considered as the output relation of the external atom. The extension of the output relation of an external atom depends on the input list, and on the definition of the external predicate $\&rdf$. Actually, the definitions of external predicates such as $\&rdf$, are associated with computable functions that take an input list $l_1, \ldots, l_n$ and an interpretation I, and return an output relation. The availability of I as input value makes relational extensions of predicates accessible to external atoms. Usually the names of predicates whose extensions are accessed are mentioned in the input list, together with other input values. Consider for instance the atom $\&reach[knows, john](X)$: the predicate $\&reach$ might be defined in a way such that $\&reach[knows, john](x)$ evaluates to true for any x which is reachable from $john$ through the current extension of the binary predicate $knows$.

We now give a more formal overview of this simple, yet powerful framework.

$$\mathit{subRelation}(\mathit{brotherOf}, \mathit{relativeOf}). \quad (4)$$

$$\mathit{brotherOf}(\mathit{john}, \mathit{al}). \quad \mathit{relativeOf}(\mathit{john}, \mathit{joe}). \quad \mathit{brotherOf}(\mathit{al}, \mathit{mick}). \quad (5)$$

$$\mathit{invites}(\mathit{john}, X) \vee \mathit{skip}(X) \leftarrow X \neq \mathit{john}, \&\mathit{reach}[\mathit{relativeOf}, \mathit{john}](X). \quad (6)$$

$$R(X, Y) \leftarrow \mathit{subRelation}(P, R), P(X, Y). \quad (7)$$

$$\mathit{someInvited} \leftarrow \mathit{invites}(\mathit{john}, X). \quad (8)$$

$$\leftarrow \text{not } \mathit{someInvited}. \quad (9)$$

$$\leftarrow \&\mathit{degs}[\mathit{invites}](\mathit{Min}, \mathit{Max}), \mathit{Max} > 2. \quad (10)$$

Fig. 1. Example HEX-program

3.2 Formal Concepts

Syntax of HEX-Programs. The vocabulary Φ comprises besides $\mathcal{C}$ and $\mathcal{X}$ also *external predicate names* $\mathcal{G}$, which are prefixed with "&". We note that constant symbols serve both as individual and predicate symbols (no $\mathcal{P}$ is needed).

A *higher-order atom* (or *atom*) is a tuple $(Y_0, Y_1, \ldots, Y_n)$, where $Y_0, \ldots, Y_n$ are terms and $n \geq 0$. Intuitively, Y_0 is the predicate name, thus we use the familiar notation $Y_0(Y_1, \ldots, Y_n)$. The atom is *ordinary*, if Y_0 is a constant. For example, $(x, \mathit{rdf{:}type}, c)$, $\mathit{node}(X)$, and $D(a, b)$, are atoms; the first two are ordinary.

An *external atom* is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m) \ , \quad (3)$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms (called *input* and *output* lists, respectively), and $\&g \in \mathcal{G}$ is an external predicate name. We assume that $\&g$ has fixed lengths $in(\&g) = n$ and $out(\&g) = m$ for input and output lists, respectively. An external atom provides a way for deciding the truth value of an output tuple depending on the extension of a set of input predicates: in this respect, an external predicate $\&g$ is equipped with a function $f_{\&g}$ evaluating to true for proper input values.

A HEX-*rule* r is of the form (1), where all α_i are (higher-order) atoms and each β_j is a (higher-order) atom or an external atom; strong negation is disregarded. $H(r)$, $B(r)$, $B^+(r)$, and $B^-(r)$ are as in Section 2; r is *ordinary*, if it contains only ordinary atoms.

Definition 3. *A* HEX-program *is a finite set P of* HEX-*rules. It is* ordinary*, if all rules are ordinary.*

Example 3 ([17]). Consider the HEX-program P in Figure 1. Informally, this program randomly selects a certain number of John's relatives for invitation. The first line states that $\mathit{brotherOf}$ is a subrelation of $\mathit{relativeOf}$, and the next line gives concrete facts. The disjunctive rule (6) chooses relatives, employing the external predicate $\&\mathit{reach}$. This latter predicate takes in input a binary relation e and a node name n, returning the nodes reachable from n when traversing the graph described by e (see the following Example 5). Rule (7) axiomatizes subrelation inclusion exploiting higher-order atoms; that is, for those couples of binary predicates p, r for which it holds $\mathit{subRelation}(p, r)$, it must be the case that $r(x, y)$ holds whenever $p(x, y)$ is true.

The constraints (9) and (10) ensure that the number of invitees is between 1 and 2, using (for illustration) an external predicate $\&\,degs$ from a graph library. Such a predicate has a valuation function $f_{\&\,degs}$ where $f_{\&degs}(I, e, min, max)$ is true iff min and max are, respectively, the minimum and maximum vertex degree of the graph induced by the edges contained in the extension of predicate e in interpretation I.

Semantics of HEX-Programs. In the sequel, let P be a HEX-program. As for ordinary programs, unless specified otherwise, $\mathcal{C}$ and $\mathcal{G}$ are implicitly given by P. The *Herbrand base* of P, denoted HB_P, is the set of all ground atoms and external atoms (we disregard here negative literals). The grounding of a rule r, $grnd(r)$, and of a program P, $grnd(P)$, are analog as above.

Example 4 ([17]). Given $\mathcal{C} = \{edge, arc, a, b\}$, ground instances of $E(X, b)$ are, e.g., $edge(a, b)$, $arc(a, b)$, $a(edge, b)$, and $arc(arc, b)$. The ground instances of the external atom $\&\,reach[edge, N](X)$ are all possible combinations where N and X are replaced by elements from $\mathcal{C}$; some examples are $\&\,reach[edge, edge](a)$, $\&\,reach[edge, arc](b)$, and $\&\,reach[edge, edge](edge)$.

An *interpretation relative to* P is any subset $I \subseteq HB_P$ containing only atoms. The notion of satisfaction (model) of rules and programs relative to I is defined as in Section 2, using for ground higher-order atoms and external atoms the following clauses:

- I satisfies a ground higher-order atom $a \in HB_P$ ($I \models a$) iff $a \in I$.
- I satisfies a ground external atom $a = \&\,g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ ($I \models a$) iff $f_{\&g}(I, y_1, \ldots, y_n, x_1, \ldots, x_m) = 1$, where $f_{\&g}$ is a (fixed) $(n+m+1)$-ary Boolean function associated with $\&\,g \in \mathcal{G}$ that assigns each tuple $(I, y_1 \ldots, y_n, x_1, \ldots, x_m)$ either 0 or 1, where $n = in(\&\,g)$, $m = out(\&\,g)$, $I \subseteq HB_P$, and $x_i, y_j \in \mathcal{C}$.

Example 5 ([17]). Let us associate with the external atom $\&\,reach$ a function $f_{\&reach}$ such that $f_{\&reach}(I, E, A, B) = 1$ iff B is reachable in the graph E from A. Let $I = \{e(b, c), e(c, d)\}$. Then, $I \models \&\,reach[e, b](d)$ since $f_{\&reach}(I, e, b, d) = 1$.

Note that in contrast to the semantics of higher-order atoms, which in essence reduces to first-order logic as customary (cf. [37]), the semantics of external atoms is in spirit of second order logic since it involves predicate extensions.

A HEX-program P is *satisfiable*, if it has some model. Carrying the definition of FLP-reduct fP^I from Section 2 over naturally, we then have:

Definition 4. *$I \subseteq HB_P$ is an* answer set *of a* HEX*-program P iff I is a minimal model of fP^I.*

Considering example 3, as John's relatives are determined to be Al, Joe, and Mick, P has six answer sets, each of which contains one or two of the facts $invites(john, al)$, $invites(john, joe)$, and $invites(john, mick)$.

In principle, the truth value of an external atom depends on its input and output lists and on the entire model of the program. In practice, however, we can identify certain types of input terms that allow to restrict the input interpretation to specific relations. The Boolean function associated with the external atom $\&\,reach[edge, a](X)$ for instance will only consider the extension of the predicate $edge$ and the constant value a for computing its result, and simply ignore everything else of the given input interpretation.

An important property of answer sets, which is guaranteed by the use of the FLP-reduct, is groundedness.

Proposition 2 ([17]). *Every answer set of a* HEX*-program P is a minimal model of P.*

This would not be generally the case if instead of fP^I we would use the GL-reduct from Section 2; however, it is if all external atoms α in P are *monotonic*, i.e., whenever $I \subseteq J \subseteq HB_P$ and $I \models \alpha'$ for a ground instance α' of α, then $J \models \alpha'$. Then the following result, which generalizes Proposition 1, can be easily shown. Let $ans_{FLP}(P)$ and $ans_{GL}(P)$ denote the answer sets of P defined using the FLP-reduct and the GL-reduct, respectively.

Theorem 1. *Suppose P is a* HEX*-program such that all external atoms in P are monotonic. Then $ans_{GL}(P) = ans_{FLP}(P)$.*

3.3 Evaluation of HEX-Programs

Some concerns might be raised regarding practical evaluation of HEX-programs. Arguably, the features of HEX-programs (mainly, the possibility of combining higher-order constructs with external atoms, with no restriction on their usage) enforce some design constraint that would compromise the practical adoption of this formalism in its full generality. To this end, although keeping desirable advantages, feasible classes of HEX-programs for implementation were identified in [18], together with a general method for combining and evaluating sub-programs belonging to arbitrary classes, thus enlarging the variety of programs whose execution is practicable. As detailed in [38] HEX-programs can be evaluated by means of calls to a traditional answer set solver, interleaved with calls to external atom functions. The evaluation order is given by means of a generalization of the splitting sets method [30]. A recently explored way to further improve evaluation is program decomposition, by exploiting independence information of the external atoms, which is used to restrict the evaluation domain in each decomposed program [12].

3.4 Implementation and Applications

HEX-programs have been implemented within the dlvhex prototype,[2] which is based on a flexible and modular architecture. The evaluation of the external atoms is realized by plugins, which are loaded at run-time. Third-party developers can easily contribute by adding new external predicates to the (rich) pool of available external predicates.

HEX-programs have been deployed to a number of applications in different contexts, of which we mention some here. Hoehndorf et al. [28] showed how to combine multiple biomedical upper ontologies by extending the first-order semantics of terminological knowledge with default logic. The corresponding prototype implementation of such kind of system is given by mapping the default rules to a HEX-program. Fuzzy

[2] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

extensions of answer-set programs in relation with HEX-programs are given in [33,27]. While [33] maps fuzzy answer set programs to HEX-programs, [27] defines a fuzzy semantics for HEX-programs and gives a translation to standard HEX-programs. In [34], the planning language $\mathcal{K}^c$ was introduced which features external function calls in spirit of HEX-programs. Also, HEX-programs have been applied to optimal credential selection in the context of trust negotiation processes [38].

4 Modular Nonmonotonic Logic Programming

We now turn to Modular Nonmonotonic Logic Programs (MLPs) [7], which have their roots in *Logic Programs with Generalized Quantifiers* (GQLPs) [13] and HEX-programs. GQLPs extend logic programs by *generalized quantifiers (GQs)*, i.e., formulas $Q\boldsymbol{x}[R(\boldsymbol{x})]$ with generalized quantifier $Q\boldsymbol{x}$ over a structure defined by the relation R (cf. [40] for background). For instance, for the transitive closure GQ Q_{tc}, the rule

$$t(X,Y) \leftarrow Q_{tc}[e](X,Y) \tag{11}$$

sets t to the transitive closure of the binary relation defined by e. Naturally, we may view GQs as interfaces to logic programs; thus Q_{tc}, may be defined as the logic program

$$tc(X,Y) \leftarrow e(X,Y). \tag{12}$$
$$tc(X,Y) \leftarrow tc(X,Z), tc(Z,Y). \tag{13}$$

with "input" predicate e and "output" predicate tc. Then, (11) may be seen as a module that calls a submodule defined by (12) and (13). Following this line, GQLPs can be used as a host to define a semantics for modular logic programs.

In [7], the modular logic programs allow for representing disjunctive logic programs in modules, which can use module atoms to access and update knowledge in other logic programs. Module atoms can be seen as an abstract way to interface with other programs, since the update mechanism of this kind of atoms gives rise to multiple instances of logic programs. HEX-programs share this similarity of updating external knowledge sources, but, unlike MLPs, these updates play only a role "locally," while updates in MLPs have the potential to trigger the creation of new "module instances," which act as new "global" entities. We will reconsider this issue later in this section. Next, we compare MLPs with HEX-programs and GQLPs.

MLP vs. GQLPs and HEX-Programs. The first stepping stone towards modular logic programs were GQLPs, which are programs that have besides standard literals also generalized quantifier literals in the body of rules. On top of that, the interface to the modules of a modular logic program can be conceived as GQs.

This approach has been enhanced in HEX-programs which use the FLP-reduct to deal with negation-as-failure, and have disjunctive heads. Essentially, HEX-programs are similar to GQLPs, and external atoms are in the same vein as GQLPs.

Both formalisms have limitations. In GQLPs, only *hierarchical* modular logic programs were defined, i.e., programs whose subprograms do not refer back to the calling program. If one defines in a HEX-program external atoms as interfaces to logic program modules, this restriction is not explicit; however, there is an implicit understanding that external sources are independent of the calling HEX-program, and thus that modules are acyclic. Hence, the first problem worth to overcome is the acyclic module topology, and to define a semantics that can deal with arbitrarily intertwined modules, where each of them can call each other (or themselves), possibly in a recursive way.

A second shortcoming of GQLPs is that their answer sets lack *groundedness*. E.g.,

$$P = \{\ p(a) \leftarrow C_{\forall}[p]\ \}$$

has two answer sets, viz. $M_1 = \emptyset$ and $M_2 = \{p(a)\}$. While M_1 is a minimal model of P (and thus intuitively grounded), M_2 is not; hence, answer sets of GQLPs may be "unfounded." This anomaly is due to the use of a Gelfond-Lifschitz style reduct that treats external atoms like not-literals; in this way, self-supporting beliefs are possible, similar as in Autoepistemic Logic (AEL); indeed, P paraphrases the canonical AEL theory $T = \{Lp(a) \supset p(a)\}$ that has two stable expansions akin to M_1 and M_2 (cf. [31]). Similarly, cyclic logic program modules based on HEX-programs lack groundedness (the above example is easily recast to this setting using two cyclic modules).

The above shortcomings are remedied in MLPs: they impose no restriction on calls in a program and allow for modules that may recursively access other modules; unfounded answer sets are prevented by using the FLP-reduct, which ensures minimality of answer sets. Furthermore, taking into account that modules are parts of a global program, MLPs have a global-state semantics in which Pareto-optimal states are singled out.

4.1 Formal Concepts

Modular logic programs (MLPs) consist of modules as a means to structure logic programs. The modules allow for input provided by other modules, through call by value, and may call each other in (mutual) recursion. We illustrate this on an example.

Example 6. Suppose we have three modules named $P_1[]$, $P_2[q_2]$, and $P_3[q_3]$ with rules $R_1 = \{q(a).\ \ q(b).\ \ ok \leftarrow P_2[q].even.\}$,

$$R_2 = \left\{\begin{array}{c} q_2'(X) \vee q_2'(Y) \leftarrow q_2(X), q_2(Y), \\ \qquad\qquad X \neq Y. \\ skip_2 \leftarrow q_2(X), \mathrm{not}\ q_2'(X). \\ even \leftarrow \mathrm{not}\ skip_2. \\ even \leftarrow skip_2, P_3[q_2'].odd. \end{array}\right\},\ R_3 = \left\{\begin{array}{c} q_3'(X) \vee q_3'(Y) \leftarrow q_3(X), q_3(Y), \\ \qquad\qquad X \neq Y. \\ skip_3 \leftarrow q_3(X), \mathrm{not}\ q_3'(X). \\ odd \leftarrow skip_3, P_2[q_3'].even. \end{array}\right\}$$

respectively. Informally, ok is computed true in P_1, if P_2 (having formal parameter q_2) computes $even$ true on input of predicate q's value. P_2 does so in mutual recursion with P_3 (having formal parameter q_3), which computes odd; for this, they compute for the recursive call in q_i' the input q_i minus one randomly removed element (cf. Example 2).

Syntax of MLPs. The vocabulary Φ also has a set $\mathcal{M}$ of *module names* P with fixed associated lists $\boldsymbol{q} = q_1, \ldots, q_k$ ($k \geq 0$) of predicate names $q_i \in \mathcal{P}$ (the formal input parameters), denoted $P[\boldsymbol{q}]$; function symbols are disregarded.

Ordinary atoms (simply atoms) have the form $p(\boldsymbol{t})$, where $p \in \mathcal{P}$ has arity $n \geq 0$ and $\boldsymbol{t} = t_1, \ldots, t_n$ are terms. A *module atom* has the form

$$P[p_1, \ldots, p_k].o(t_1, \ldots, t_l) \ , \tag{14}$$

where (i) $P \in \mathcal{M}$ with $P[q_1, \ldots, q_k]$, (ii) $p_1, \ldots, p_k$ is an input list of predicate names $p_i \in \mathcal{P}$ matching the arity of q_i, and (iii) $o(t_1, \ldots, t_l)$ is an ordinary atom (with $o \in \mathcal{P}$). Intuitively, a module atom provides a way for deciding the truth value of a (ground) atom $o()$ in a program P depending on a set of input predicates.

An *MLP-rule* r is of the form (1), where all α_i are atoms, each β_j is an atom or a module atom, and $k \geq 1$;[3] r is *ordinary*, if it contains only ordinary atoms.

A *module* $m = (P, R)$ consists of a module name $P \in \mathcal{M}$ and a finite set R of rules. *Main modules* have no input (i.e., have $P[\,]$), while *library modules* have arbitrary input. As usual, empty input $[\,]$ and argument lists $(\,)$ are omitted.

Definition 5. *A* modular logic program *(MLP) is of the form* $\mathbf{P} = (m_1, \ldots, m_n)$*,* $n \geq 1$*, where all* $m_i = (P_i, R_i)$ *are modules and at least one* m_i *is a main module.*

To have no unused modules, it is assumed that $\mathcal{M} = \{P_1, \ldots, P_n\}$. $\mathbf{P}$ is *ground*, iff each module M_i is *ground*, which means that all rules in R_i are ground.

The *call graph of an MLP* $\mathbf{P}$ is a labeled digraph $CG_{\mathbf{P}} = (V, E, l)$ with vertex set $V = VC(\mathbf{P})$ and an edge e from $P_i[S]$ to $P_k[T]$ in E iff $P_k[\boldsymbol{p}].o(\boldsymbol{t})$ occurs in $R(m_i)$; furthermore, e is labeled with an input list $\boldsymbol{p}$, denoted $l(e)$.

Example 7. The MLP in Example 6 consists of three modules $\mathbf{P} = (m_1, m_2, m_3)$, where $m_1 = (P_1, R_1)$ is the main module and $M_i = (P_i[q_i], R_i)$, $i = 2, 3$ are library modules. Furthermore, m_1 is ground while m_2 and m_3 are not.

Let $S^i_\emptyset = \emptyset$, $S^i_a = \{q_i(a)\}$, $S^i_b = \{q_i(b)\}$, and $S^i_{ab} = \{q_i(a),\ q_i(b)\}$. Then $VC(\mathbf{P}) = \{P_1[\emptyset], P_2[S^2_v], P_3[S^3_w]\}$, where $v, w \in \{\emptyset, a, b, ab\}$, and $CG_{\mathbf{P}}$ has edges $P_1[\emptyset] \xrightarrow{q} P_2[S^2_v]$, $P_2[S^2_v] \xrightarrow{q'_2} P_3[S^3_w]$, and $P_3[S^3_w] \xrightarrow{q'_3} P_2[S^2_v]$.

Semantics of MLPs. The semantics of MLPs is given in terms of grounding and Herbrand interpretations customary in logic programming. Naturally, also modules $(P[\boldsymbol{q}], R)$ must be instantiated before they can be "used;" there is one instance per possible input for $\boldsymbol{q}$ (referred to as *value call*). To focus on "relevant" module instances, the call chain and an embracing context of value calls are considered, while others are (in essence) ignored.

The *Herbrand base* of an MLP $\mathbf{P}$ (implicitly defining $\Phi = \Phi_{\mathbf{P}}$) is the set $HB_{\mathbf{P}}$ of all ground ordinary and module atoms from vocabulary Φ. The grounding $grnd(r)$ of a rule and $grnd(R)$ of a rule set R are as usual; the grounding of a module $m = (P[\boldsymbol{q}], R)$ is $grnd(m) = (P[\boldsymbol{q}], grnd(R))$, and the grounding of an MLP $\mathbf{P} = (m_1, \ldots, m_n)$ is $gr(\mathbf{P}) = (grnd(m_1), \ldots, grnd(m_n))$.

To define module instances, we need the following notations. For any set S of ground atoms and lists $\boldsymbol{p} = p_1, \ldots, p_k$ and $\boldsymbol{q} = q_1, \ldots, q_k$ of predicate names, let $S|_{\boldsymbol{p}} = \bigcup_{i=1}^k \{p_i(\boldsymbol{c}) \in S\}$ and $S|^{\boldsymbol{q}}_{\boldsymbol{p}} = \bigcup_{i=1}^k \{q_i(\boldsymbol{c}) \mid p_i(\boldsymbol{c}) \in S\}$.

[3] Constraints $\leftarrow B(r)$ (banned for satisfiability) are easily emulated with $f \leftarrow \text{not } f, B(r)$.

Then, for a module $m_i = (P_i[\boldsymbol{q_i}], R_i)$, a *value call with input* S is a pair (P_i, S) where $S \subseteq HB_{\mathbf{P}}|_{\boldsymbol{q_i}}$, also written as $P_i[S]$; its *instantiation with* S is the rule set $I_{\mathbf{P}}(P_i[S]) = R_i \cup S$. The possible instances of all modules m_i in $\mathbf{P}$ are naturally indexed by the set $VC(\mathbf{P})$ of all possible $P_i[S]$. Technically, they form an (indexed) tuple $I(\mathbf{P}) = (I_{\mathbf{P}}(P_i[S]) \mid P_i[S] \in VC(\mathbf{P}))$ called the *instantiation* of $\mathbf{P}$. The latter is a *rule base*, which are tuples $\mathbf{R} = (R_{P_i[S]} \mid P_i[S] \in VC(\mathbf{P}))$ of rule sets $R_{P_i[S]}$.

An *interpretation* $\mathbf{M}$ is now an (indexed) tuple $(M_i/S \mid P_i[S] \in VC(\mathbf{P}))$ of sets M_i/S of ordinary ground atoms. At a value call $P_i[S]$, it satisfies (is a model of)

- a ground atom $\alpha \in HB_{\mathbf{P}}$, denoted $\mathbf{M}, P_i[S] \models \alpha$, iff (i) $\alpha \in M_i/S$ when α is ordinary, and (ii) $o(\boldsymbol{c}) \in M_k/((M_i/S)|^{\boldsymbol{q_k}}_{\boldsymbol{p}})$, when $\alpha = P_k[\boldsymbol{p}].o(\boldsymbol{c})$ is a module atom;
- a ground rule r ($\mathbf{M}, P_i[S] \models r$), iff $\mathbf{M}, P_i[S] \models H(r)$ or $\mathbf{M}, P_i[S] \not\models B(r)$, where (i) $\mathbf{M}, P_i[S] \models H(r)$, iff $\mathbf{M}, P_i[S] \models \alpha$ for some $\alpha \in H(r)$, and (ii) $\mathbf{M}, P_i[S] \models B(r)$, iff $\mathbf{M}, P_i[S] \models \alpha$ for all $\alpha \in B^+(r)$ and $\mathbf{M}, P_i[S] \not\models \alpha$ for all $\alpha \in B^-(r)$;
- a set of ground rules R ($\mathbf{M}, P_i[S] \models R$) iff $\mathbf{M}, P_i[S] \models r$ for all $r \in R$.

Furthermore, $\mathbf{M}$ satisfies a rule base $\mathbf{R}$ ($\mathbf{M} \models \mathbf{R}$), if $grnd(R_{P_i[S]})$ at all $P_i[S]$ are satisfied by $\mathbf{M}$, and $\mathbf{M}$ satisfies $\mathbf{P}$ ($\mathbf{M} \models \mathbf{P}$), if $\mathbf{M} \models I(grnd(\mathbf{P}))$.

To focus on relevant module instances w.r.t. an interpretation $\mathbf{M}$, we use the *relevant call graph* $CG_{\mathbf{P}}(\mathbf{M})$ *of* $\mathbf{P}$, which is the subgraph of $CG_{\mathbf{P}}$ containing all edges $e : P_i[S] \overset{l(e)}{\rightarrow} P_k[T]$ in $CG_{\mathbf{P}}$ such that $(M_i/S)|^{\boldsymbol{q_k}}_{l(e)} = T$, with nodes induced by the edges plus all main module instantiations (called *relevant instances* w.r.t. $\mathbf{M}$).

Example 8. For the interpretation $\mathbf{M}$ such that $M_1/\emptyset = \{q(a), q(b), ok\}$, $M_2/S^2_{ab} = \{q_2(a),\ q_2(b), q_2'(a), skip_2, even\}$, $M_2/\emptyset = \{even\}$, and $M_3/S^3_a = \{q_3(a), skip_3, odd\}$, the nodes of $CG_{\mathbf{P}}(\mathbf{M})$ are $P_1[\emptyset]$, $P_2[S^2_{ab}]$, $P_2[\emptyset]$, and $P_3[S^3_a]$.

The nodes of $CG_{\mathbf{P}}(\mathbf{M})$ are the smallest set of module instances which is intuitively involved in building an answer set. As an over-approximation, a superset C of these nodes, called *context*, is used in [7]; we omit this here for simplicity.

To define answer sets, we first need minimal models, which are given as follows: let $\mathbf{M} \leq \mathbf{M}'$ iff $M_i/S \subseteq M_i'/S$ for all $P_i[S]$. Then a model $\mathbf{M}$ of $\mathbf{P}$ (resp., a rule base $\mathbf{R}$) is *minimal*, if $\mathbf{P}$ (resp., $\mathbf{R}$) has no model $\mathbf{M}' \neq \mathbf{M}$ such that $\mathbf{M}' \leq \mathbf{M}$.

Now the FLP-reduct is generalized to work on MLPs componentwise where module instantiations outside the relevant call graph are not touched. Formally, the *reduct* $f\mathbf{P}(P_i[S])^{\mathbf{M}}$ *of* $\mathbf{P}$ *at* $P_i[S]$ *w.r.t.* $\mathbf{M}$ is (i) the FLP-reduct $fI_{gr(\mathbf{P})}(P_i[S])^{M_i/S}$, i.e., $\{r \in I_{gr(\mathbf{P})}(P_i[S]) \mid \mathbf{M}, P_i[S] \models B(r)\}$, if $P_i[S]$ is in $CG_{\mathbf{P}}(\mathbf{M})$, and (ii) $I_{gr(\mathbf{P})}(P_i[S])$ otherwise. The *reduct of* $\mathbf{P}$ *w.r.t.* $\mathbf{M}$ is $f\mathbf{P}^{\mathbf{M}} = (f\mathbf{P}(P_i[S])^{\mathbf{M}} \mid P_i[S] \in VC(\mathbf{P}))$.

Definition 6. *An interpretation* $\mathbf{M}$ *of an MLP* $\mathbf{P}$ *is an* answer set of $\mathbf{P}$*, iff* $\mathbf{M}$ *is a minimal model of* $f\mathbf{P}^{\mathbf{M}}$.

Example 9. Recall interpretations of the form $\mathbf{M}$ from Example 8. It is easily verified that for every node $P_i[S]$ in $CG_{\mathbf{P}}(\mathbf{M})$, the respective interpretation M_i/S is minimal for $f\mathbf{P}(P_i[S])^{\mathbf{M}}$. Therefore, any such $\mathbf{M}$ is an answer set of $\mathbf{P}$ iff for every $P_i[S]$ outside $CG_{\mathbf{P}}(\mathbf{M})$, the interpretation M_i/S is a minimal model of $I_{gr(\mathbf{P})}(P_i[S])$.

4.2 Semantic Properties of MLPs

MLPs conservatively extend ordinary logic programs, and many of the nice semantic properties of the latter generalize to them. We recall below a couple of them from [7].

Proposition 3 ([7]). *Let R be an ordinary logic program. Then M is an answer set of R iff $\mathbf{M} = (M_1/\emptyset)$ with $M_1/\emptyset = M$ is an answer set of the MLP (m_1), where $m_1 = (P_1[\,], R)$ is a main module and P_1 is a module name.*

An important observation is that the answer sets of an MLP $\mathbf{P}$ are grounded; this is due to the use of the FLP-reduct (a GL-style reduct would behave differently).

Proposition 4. *Every answer set of $\mathbf{P}$ is a minimal model of $\mathbf{P}$.*

Moreover, in absence of negation-as-failure also the converse holds.

Proposition 5. *The answer sets of a positive MLP $\mathbf{P}$ coincide with its minimal models.*

For a suitable notion of intersection, we get that the models of a Horn MLP $\mathbf{P}$ are closed under intersection; hence, a Horn MLP has a canonical answer set.

Proposition 6. *Every Horn MLP $\mathbf{P}$ has a single answer set, which coincides with its least model.*

4.3 Computation

Exploiting Proposition 6, answer sets of Horn MLPs can be computed by means of a bottom up fixed-point computation. However, this is not effective, as many irrelevant module instantiations might be considered that do not contribute to the part of interest, given by the main modules. As has been shown in [7], in the general case, one has to deal with double exponential many instantiations, which is clearly infeasible in practice. This calls for refined methods that overcome the need to instantiate all possible modules.

Further work [8] addresses efficient evaluation of MLPs using a generalization of the splitting sets method [30] that takes relevance information into account. For a certain subclass of MLPs that obeys a notion of *call stratification* and *input stratification*, we have an algorithm that evaluates MLPs top-down, more importantly, expands only relevant instantiations during the evaluation, which speeds up the process.

5 Multi-context Systems

In this section, we turn to another nonmonotonic formalism that provides access to external sources in the realm of context-based reasoning. Informally, a multi-context system describes the information available in a number of contexts (e.g., to different agents or views) and specifies the information flow between those contexts. Furthering work in [32,25], the Trento School developed monotonic heterogeneous multi-context systems [26] with the aim to integrate different inference systems; informally, they viewed contexts as pairs $Context_i = (\{T_i\}, \Delta_{br})$ where each $T_i = (L_i, \Omega_i, \Delta_i)$ is a formal system, and Δ_{br} consists of *bridge rules* of the form

$$(c_1\, p_1), \ldots, (c_k\, p_k) \Rightarrow (c_j\, q_j)$$

using labeled formulas $(c\, p)$ where p is from the language L_c. Giunchiglia and Serafini gave a collection of such contexts a semantics in terms of local models plus compatibility conditions, which respects information flow across contexts via bridge rules. Noticeably, reasoning within/across contexts is monotonic.

Brewka et al. [4] extended the framework to Contextual Default Logic (CDL), improving on [36], where bridge rules with negation were considered. CDL integrates nonmonotonic inference systems of the same kind, viz. theories in Reiter's Default Logic. Here, defaults may refer to other contexts and play the role of bridge rules.

The Multi-Context Systems (MCS) of [2] generalized these approaches, by accommodating *heterogeneous* and both *monotonic* and *nonmonotonic* contexts, thus capable of integrating "typical" monotonic KR logics like description logics or temporal logics, and nonmonotonic logics like Reiter's Default Logic, Answer Set Programming, circumscription, defeasible logic, or theories in autoepistemic logic; in several of the latter, a knowledge base gives rise to multiple belief sets in general. In our taxonomy, MCSs have a "global-state" semantics, that is defined via bridge rules and follows the classical ASP definition, extended to this setting. Before we present MCSs in more detail, it is helpful to compare them to MLPs and HEX-programs.

MCSs vs. MLPs and HEX-Programs. Compared to MLPs, MCSs are more general since the contexts (viewed as modules) consist of general reasoning systems or logics, respectively, and not only of ASP programs. The MCS semantics is similar in spirit to the semantics of MLPs in [13], but global-state rather than local-state, and quite different from MLPs in [7], which are global-state and use FLP-reduct.

For HEX-programs, the comparison shows a more complex picture:

– MCSs are similar to HEX-programs where the external sources are knowledge bases.
– We may view MCSs as a more general, hybrid formalism than such HEX-programs, in which bridge rules for the information flow are distinguished (at the meta-level) from the formulas of the knowledge base; HEX-rules with external atoms can be seen as bridge rules, and HEX-rules without as rules of the knowledge base describing the local state.
– HEX-programs are local-state and use the FLP-reduct, while MCSs are global-state and use GL-style reducts. The local-state property prevents a naive encoding of MCSs into HEX (as local belief sets of other contexts can not be directly accessed); however, under some (weak) condition, such an encoding is possible; we discuss this in Section 5.2.
– In HEX-programs, we may abstractly access knowledge bases through powerful reasoning services of an API beyond checking formula membership in local belief sets.

5.1 Formal Concepts

In [2], a "logic" is, very abstractly, a tuple $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$, where

- $\mathbf{KB}_L$ is a set of well-formed knowledge bases, each being a set (of formulas),
- $\mathbf{BS}_L$ is a set of possible belief sets, each being a set (of formulas), and
- $\mathbf{ACC}_L : \mathbf{KB}_L \rightarrow 2^{\mathbf{BS}_L}$ assigns each $kb \in \mathbf{KB}_L$ a set of acceptable belief sets;

L is *monotonic*, if $\mathbf{ACC}_L$ assigns each kb a single belief set (denoted S_{kb}), and $kb \subseteq kb'$ implies $S_{kb} \subseteq S_{kb'}$. We can think of knowledge bases as logic programs, classical theories etc; the possible belief sets are those which are syntactically admissible (e.g., deductively closed sets of sentences, set of literals, etc); and $\mathbf{ACC}_L$ respects that a knowledge base might have one, multiple, or even no acceptable belief set in the logic.

Access to other contexts is facilitated via bridge rules for heterogenous logics. Given logics $L = L_1, \ldots, L_n$, an L_i-bridge rule over L, $1 \leq i \leq n$, is of the form

$$s \leftarrow (r_1 : p_1), \ldots, (r_j : p_j), \mathbf{not}\ (r_{j+1} : p_{j+1}), \ldots, \mathbf{not}\ (r_m : p_m) \tag{15}$$

where $r_k \in \{1 \ldots, n\}$ and p_k is an element of some belief set of L_{r_k}, $1 \leq k \leq m$, and $kb \cup \{s\} \in \mathbf{KB}_i$ for each $kb \in \mathbf{KB}_i$.

Multi-context systems are then defined as follows.

Definition 7. *A* multi-context system $M = (C_1, \ldots, C_n)$ *consists of contexts* $C_i = (L_i, kb_i, br_i)$*, where* $L_i = (\boldsymbol{KB}_i, \boldsymbol{BS}_i, \boldsymbol{ACC}_i)$ *is a logic,* $kb_i \in \boldsymbol{KB}_i$ *is a knowledge base, and* br_i *is a set of* L_i*-bridge rules over* $L = L_1, \ldots, L_n$*,* $1 \leq i \leq n$*.*

Example 10. As a simple example, we consider $M = (C_1, C_2)$, where the contexts are different views of a paper by its co-authors A_1 and A_2 who reason in different logics. In C_1, we have Classical Logic as L_1, the knowledge base $kb_1 = \{\, \mathit{unhappy} \supset \mathit{revision} \,\}$, and the bridge rules $br_1 = \{\, \mathit{unhappy} \leftarrow (2 : \mathit{work}) \,\}$. Intuitively, if A_1 is unhappy about the paper, then she wants a revision, and if A_2 finds that the paper needs more work, then A_1 feels unhappy. In C_2, we have Answer Set Programming as L_2, the knowledge base $kb_2 = \{\, \mathit{accepted} \leftarrow \mathit{good}, \text{not } \neg \mathit{accepted} \,\}$ and bridge rules $br_2 = \{\, \mathit{work} \leftarrow (1 : \mathit{revision}); \mathit{good} \leftarrow \mathbf{not}\ (1 : \mathit{unhappy})\}$. Intuitively, A_2 thinks that the paper, if good, is usually accepted; moreover, she infers that more work is needed if A_1 wants a revision, and that the paper is good if there is no evidence that A_1 is unhappy.

The semantics of an MCS is defined in terms of special belief states, which are sequences $S = (S_1, \ldots, S_n)$ such that each S_i is an element of $\mathbf{BS}_i$. Intuitively, S_i should be a belief set of the knowledge base kb_i; however, also the bridge rules must be respected; to this end, kb_i is augmented with the conclusions of its bridge rules that are applicable. More precisely, a bridge rule r of form (15) is *applicable in* S, if $p_i \in S_{r_i}$, for $1 \leq i \leq j$, and $p_k \notin S_{r_k}$, for $j + 1 \leq k \leq m$. Denote by $\mathit{head}(r)$ the head of r and by $\mathit{app}(R, S)$ the set of bridge rules $r \in R$ that are applicable in S. Then,

Definition 8. *A belief state* $S = (S_1, \ldots, S_n)$ *of a multi-context system* M *is an* equilibrium *iff* $S_i \in \boldsymbol{ACC}_i(kb_i \cup \{\mathit{head}(r) \mid r \in \mathit{app}(br_i, S)\})$*,* $1 \leq i \leq n$*.*

An equilibrium thus is a belief state which contains for each context an acceptable belief set, given the belief sets of the other contexts.

Example 11 (ctd). Reconsidering $M = (C_1, C_2)$ from Example 10, we find that M has two equilibria, viz.

– $E_1 = (Cn(\{\mathit{unhappy}, \mathit{revision}\}), \{\mathit{work}\})$ and
– $E_2 = (Cn(\{\mathit{unhappy} \supset \mathit{revision}\}), \{\mathit{good}, \mathit{accepted}\})$,

where $Cn(\cdot)$ is the set of all classical consequences. As for E_1, the bridge rule of C_1 is applicable in E_1, and $Cn(\{\mathit{unhappy}, \mathit{revision}\})$ is the (single) acceptable belief set of $kb_i \cup \{\mathit{unhappy}\}$; the first bridge rule of C_2 is applicable in E_1, but not the second; clearly, $\{\mathit{work}\}$ is the single answer set of $kb_2 \cup \{\, \mathit{work} \,\}$.

As for E_2, the bridge rule of C_1 is not applicable in E_1, and $Cn(\{unhappy \supset revision\} = Cn(kb_1)$; now the second bridge rule of C_2 is applicable but not the first, and $\{good, accepted\}$ is the single answer set of $kb_2 \cup \{good\}$.

The notion of equilibrium may remind of similar game-theoretic concepts, and in fact we may view each context C_i as a player in an n-person game where players choose belief sets. Assume that an outcome (i.e., belief state) $S = (S_1, \ldots, S_n)$, has for C_i reward 1 if $S_i \in \mathbf{ACC}_i(kb_i \cup \{head(r) \mid r \in app(br_i, S)\})$ and 0 otherwise. Then, it is easy to see that each equilibrium of M is a *Nash equilibrium* of this game (indeed, each player has optimal reward); on the other hand, there might be Nash equilibria that do not correspond to any equilibrium. This may happen e.g. if no acceptable belief sets are possible. For instance, the MCS $M = (C_1, C_2)$, where C_1 and C_2 are isolated answer set programs $\{a \leftarrow \text{not } a\}$, has no equilibrium, but $S = (\emptyset, \emptyset)$ is a Nash-equilibrium of the game. Clearly, if M has equilibria, then they coincide with the *Pareto-optimal* solutions of the game; under additional conditions (e.g., $\mathbf{ACC}_i(b_i \cup H_i) \neq \emptyset$ for each $H_i \subseteq \{h(r) \mid r \in br_i\}$) they coincide with the Nash equilibria.

Groundedness. Equilibria suffer, similar as the answer sets of modular logic programs in [13], from groundedness problems due to cyclic justifications. Informally, the reason is that bridge rules might be applied unfoundedly. E.g., in Example 10, $unhappy$ has in $E_1 = (Cn(\{unhappy, revision\}), Cn(\{work\}))$ only a cyclic justification: it is accepted in C_1 via the bridge rule, as $work$ is accepted in C_2; the latter is also via a bridge rule, as $revision$ is accepted in C_1 (by modus ponens from $unhappy \supset revision$ and $unhappy$). Here, the application of the bridge rules is unfounded.

Inspired by the definition of answer set semantics, [2] proposed grounded equilibria to overcome this. They are defined in terms of a GL-style reduct which transforms $M = (C_1, \ldots, C_n)$, given a belief state $S = (S_1, \ldots, S_n)$, into another MCS $M^S = (C_1^S, \ldots, C_n^S)$ that behaves monotonically, such that a unique minimal equilibrium exists; if it coincides with S, we have groundedness.

Formally, $C_i^S = (L_i, red_i(kb_i, S), br_i^S)$, where $red_i(kb_i, S)$ maps kb_i and S to a monotonic core of L_i and br_i^S is the GL-reduct of br_i w.r.t. S, i.e., contains $s \leftarrow (r_1 : p_1), \ldots, (r_j : p_j)$ for each rule of form (15) in br_i such that $p_k \notin S_{r_k}, k = j{+}1, \ldots, m$.

In addition, the following *reducibility conditions* are assumed: (i) $red_i(kb_i, S_i)$ is antimonotonic in S_i, (ii) S_i is acceptable for kb_i iff $\mathbf{ACC}_i(red_i(kb_i, S_i)) = \{S_i\}$, and (iii) $red_i(kb_i, S) \cup H_i = red_i(kb_i \cup H, S)$, for each $H_i \subseteq \{head(r) \mid r \in br_i\}$. This condition is trivially satisfied by all monotonic logics, by Reiter's Default Logic, answer set programs, etc. Grounded equilibria are then defined as follows.

Definition 9. *A belief state $S = (S_1, \ldots, S_n)$ is a grounded equilibrium of M iff S is the unique minimal equilibrium of M^S, where minimality is componentwise w.r.t. $\subseteq$.*

Example 12 (ctd.). In our review example, naturally $red(kb_i, S)$ is identity and $red(kb_2, S)$ the GL-reduct. Then E_1 is not a grounded equilibrium: M^{E_1} has the single minimal equilibrium $(Cn(\{unhappy \supset revision\}), \emptyset)) \neq E_1$. On the other hand, E_2 is a grounded equilibrium of M.

Grounded equilibria are in fact equilibria of M, as well as minimal ones. Similar as for answer sets, the grounded equilibrium of M^S can be characterized as the least fixpoint of an operator [2].

5.2 Mapping MCSs into HEX-Programs

As mentioned above, the global state view of MCSs contrasts with the local-state and inference-based view of HEX-programs, but under some condition, an MCS can be encoded into a HEX-program.

Suppose that $M = (C_1, \ldots, C_n)$ is such that in all contexts C_i, every belief set $S \in \mathbf{ACC}(kb'_i)$ has a *kernel* $\kappa(kb'_i, S) = S \cap K_i$, for some (small finite) set K_i, that uniquely identifies S, where $kb_i \subseteq kb'_i \subseteq kb_i \cup \{head(r) \mid r \in br_i\}$. (As noted in [2], the usual logics have such kernels; e.g., for Answer Set Programming, $\kappa(kb'_i, S) = S$ and K_i is the set of all ground literals.)

The equilibria of M can then be encoded by a HEX-program P_M as follows.

1. For each $p \in K_i$, set up rules
$$a_{p,i} \leftarrow \text{not}\, \bar{a}_{p,i}, \qquad \bar{a}_{p,i} \leftarrow \text{not}\, a_{p,i} \tag{16}$$
where $a_{p,i}$ $\bar{a}_{p,i}$ are fresh atoms. They guess in an interpretation I a kernel $\kappa_i^I = \{p \mid a_{p,i} \in I\}$ for some belief set of $kb_i \cup H_i^I$, where $H_i^I = \{head(r) \mid r \in br_i, a_{s,i} \in I\}$.
2. For each formula $(r_l : p_l)$ in a bridge rule $r \in br_i$ of form (15), we set up an external atom $\&con\_r_l[\,](a_{p_l})$, whose associated $f_{con\_r_l}(I, a_{p_l})$ returns 1 iff p_l is in an acceptable belief set of $kb_{r_l} \cup H_{r_l}^I$ with kernel $\kappa_{r_l}^I$.
3. We then replace each $(r_l : p_l)$ in the body of r by $\&con\_r, l[\,](a_{p_l})$, and replace the head s by the atom $a_{s,i}$. So we have
$$\begin{aligned} a_{s,i} \leftarrow\ & \&con\_r_1[\,](a_{p_1}), \ldots, \&con\_r_j[\,](a_{p_j}), \\ & \text{not}\, \&con\_r_{j+1}[\,](a_{j+1}), \ldots, \text{not}\, \&con\_r_m[\,](a_{p_m}). \end{aligned} \tag{17}$$
4. Furthermore, we add
$$b_{s,i} \leftarrow \text{not}\, a_{s,i} \tag{18}$$
$$\leftarrow \text{not}\, \&con\_r_l[\,](a_\top), \tag{19}$$
where b_s^i are fresh atoms. The rule (18) blocks minimization, while (19) eliminates an invalid guess for a kernel κ_i^I of some acceptable belief set of $kb_i \cup H_i^I$.

We then can establish the following result.

Theorem 2. *The answer sets I of P_M correspond 1-1 to the equilibria $S = (S_1, \ldots, S_n)$ of M, where each S_i is in $\mathbf{ACC}(kb_i \cup H_i^I)$ and has the kernel κ_i^I.*

Similarly, the grounded equilibria of a (reducible) M can be encoded elegantly into a HEX-program P_M^r, which results from P_M as follows:

– replace in the rules (17) $\&con\_r_l[\,](a_{p_l})$ with $\&con\_r_l{}^S[\,](a_{p_l})$, where $f_{con\_r_l{}^S}(I, a_{p_l})$ returns 1 iff p_l is in the (single) acceptable belief set of $red(kb_{r_l} \cup H_{r_l}^I, S_{r_l})$, where S_{r_l} is the acceptable belief set of $kb_{r_l} \cup H_{r_l}^I$ with kernel $\kappa_{r_l}^I$,
– drop the rules (18), and
– for each $p \in K_i$, add the constraints
$$\leftarrow \text{not}\, \&con\_i[\,](a_p), a_{p,i}. \tag{20}$$
$$\leftarrow \&con\_i[\,](a_p), \text{not}\, a_{p,i}. \tag{21}$$

Informally, they check whether the single minimal equilibrium of M^S coincides with $S = (S_1, \ldots, S_n)$, by considering the guessed kernels κ_i^I of all S_i.

For the resulting program, one can show:

Theorem 3. *The answer sets I of P_M^r correspond to 1-1 to the grounded equilibria $S = (S_1, \ldots, S_n)$ of M where each S_i is in **ACC**$(kb_i \cup H_i^I)$ and has the kernel κ_i^I.*

Refinements and alternative encodings, also for special cases, remain to be explored.

6 Ongoing Work

As demonstrated in the previous sections, combining knowledge bases with external sources based on answer set semantics is a major research focus of the KBS group at the Vienna University of Technology. Together with our external colleagues, we aim at furthering this work in several directions. Our research is driven by the general goal to develop the theoretical underpinnings for practicable and efficient implementations that serve the needs of relevant applications, as demonstrated by means of prototype implementations through experimentation and show-case applications.

Currently, we pursue the following two research projects on the topic:

- *Modular Hex programs*, funded by the Austrian Science Fund (FWF), with the goal to research and implement formalisms and reasoning techniques for providing a powerful reasoning framework in the context of modular logic programming.
- *Inconsistency Management for Knowledge-Integration Systems*, funded by the Vienna Science and Technology Fund (WWTF), with the goal to provide a general formalism and a suite of basic methods for inconsistency management in MCS, together with algorithms for their practical realization.

In the following, we summarize research issues to be addressed in these projects.

6.1 MLPs

A natural extension of MLPs is to allow program modules not only to call other modules, but also to assess external sources, i.e., ‘modules’ which are not necessarily specified as logic programs themselves. Intuitively, this is achieved by extending current MLP syntax and semantics to allow for HEX-programs in rule bases. While this is certainly of avail, also in a global view where modules are part of a ‘global program’ as an entity, optimization and relevance issues will gain importance for effective implementations. Thus, the crucial research question to address in this setting will be how to efficiently evaluate such MLPs. Some ideas to exploit generalizations of the splitting set method for restricted subclasses taking relevance information into account are briefly sketched at the end of Section 4, and initial results for the current MLP setting are reported in [8]. Further improvements, in particular when combining MLPs with HEX-programs, may be obtained by respecting further information that helps in pruning the evaluation to relevant parts. A particular case is when parts of the domain can be disregarded during the evaluation of program parts with external calls or module calls due to available independence information. Results for HEX-programs in this direction appear in [12], and experiments indicate promising improvements. An interesting related research issue is how to obtain such independence information. While it may often be easy to 'see' for the programmer, it is unclear how to proceed in a principled manner.

In a second step, we plan to carry over the MLP approach to a local-model view, with the aim to handle distributed settings appropriately, i.e., without the need of a global view for local evaluation. Semantically, this is achievable with a HEX-style state semantics for MLPs. Turning to such a distributed view will raise further challenging research issues. Concerning practical algorithms and efficient evaluation, additional characteristics of networks come into play, for instance, connection failures, scalability, or network latency. Moreover, the data traffic, i.e., the amount of data exchanged for external evaluation, must be kept low. To deal with these requirements, we envisage semantical relaxations and approximations in the vein of well-founded semantics or more general fixpoint semantics, algebraic techniques using operators, and/or multi-valued semantics.

Another aspect, which needs special attention in this setting, is the treatment of inconsistency. While in the global view, one might assume that the knowledge encoding is 'coordinated' (e.g., by a team of cooperating programmers), in a distributed setting we ought to assume that modules are created without a priori knowledge of their application, increasing the likelihood of arising conflicts. A research goal in this respect is to relax consistency requirements, in order to 'hide' conflicting information and ensure system operability. Resorting to partial models or paraconsistent reasoning techniques might be helpful. Furthermore, as mentioned earlier, incompleteness due to network errors may hinder distributed evaluation; appropriate methods are needed to cope with such situations. Techniques similar to open world reasoning might be developed for dealing with incomplete information in general, while three-valued (multi-valued) interpretations would allow to treat missing information due to network failures in an agnostic way. The `MWeb` framework [1] and [35] may give inspiration for this.

6.2 MCSs

MCSs constitute a promising approach to deal with important requirements for accessing and using data and knowledge in modern interconnected information systems, namely heterogeneity of formalisms and pointwise exchange of information rather than a central integration. However, for a practical realization, methods for adequate inconsistency handling are missing. Our research efforts address this issue at different levels. On the one hand, we are interested in applying MCS technology in order to facilitate genuine semantics of formalisms, e.g., context-based argumentation frameworks, together with ad hoc inconsistency management components. On the other hand, we aim at providing a platform for developing genuine inconsistency management of MCSs.

Argumentation. Argumentation Context Systems (ACSs) [3] specialize multi-context systems in one respect, and are more general in another. First of all, in contrast to the MCS of [2], they are homogeneous in the sense that all reasoning components in an ACS are of the same type, namely Dung style argumentation frameworks [11]. The latter are widely used as abstract models of argumentation. They are based on graphs whose nodes represent abstract arguments and edges describe attacks between arguments. Different semantics for argumentation frameworks have been defined; they specify *extensions*, i.e., subsets of the arguments which are considered jointly acceptable.

However, ACSs go beyond MCSs in two important aspects:

– The influence of an ACS module M_1 on another module M_2 can be much stronger than in an MCS. M_1 may not only provide information for M_2, it may directly affect M_2's KB and reasoning mode: M_1 may invalidate arguments or attack relationships in M_2's argumentation framework, and even determine the semantics to be used by M_2. In addition to peer-to-peer type forms of information exchange among modules, this allows one to capture hierarchical forms of argumentation as they are common in legal reasoning, where a judge may declare certain arguments as invalid, or where the type of trial requires a particular proof standard. Technically, this is achieved by an explicit representation of contexts in a genuine description language. Note that such a context is different from the usual one in MCSs: it acts as a modifier of a module's argumentation framework, and determines its semantics and reasoning mode (skeptical vs. credulous).
– A major focus in ACSs is on *inconsistency handling*. Modules are equipped with additional components called *mediators*. The main role of the mediator is to take care of inconsistencies in the information provided by connected modules. It collects the information coming in from connected modules and turns it into a consistent context for its module, using a pre-specified consistency handling method which may be based on preference information about other modules. The choice of the consistency handling method allows a broad range of scenarios to be modeled, from strictly hierarchical ones (a judge decides) to more "democratic" forms of decision making (based on voting).

We are currently investigating how to extend heterogeneous MCSs, which are not necessarily based on argumentation, in a similar fashion.

Inconsistency Management Architecture. When generalizing inconsistency management beyond specialized contexts, methods for inconsistency handling appropriate for homogenous settings cannot be utilized directly, and it is less clear how to deal with conflicts due to interaction of heterogenous knowledge bases. The objective is to abstract from techniques developed for particular formalisms with the eventual goal, to provide a general formalism and a suite of basic methods, which can be employed to declaratively specify suitable inconsistency management policies at different levels of sophistication on top of basic inconsistency management actions.

There are several research issues to achieve this goal, and different conceptual architectures can be conceived. Topologically, one may think of a hierarchical structure, where first of all each context is equipped with a local inconsistency manager, like the mediators in the above setting, acting locally and thus having access to all the information of its associated context and being capable of performing actions to ensure local consistency. This, however, will not guarantee consistency at a global level, i.e., the existence of an equilibrium for the MCS. For this purpose, a collection of contexts may agree to trust a dedicated entity to serve as their joint consistency manager, which has access to all bridge rules relevant for the contexts it is in charge of, as well as additional (but not all) information the contexts are willing to exhibit in order to resolve potential inconsistencies. Hierarchically extending this setting, with decreasing information exhibited from level to level, eventually a global inconsistency manager may take high-level decisions in order to ensure global consistency.

At each level, basic methods and algorithms for inconsistency handling need to be developed, such as for consistency checking, conflict explanation finding, conflict assessment, and methods for conflict resolution, taking into account the information that is available for the inconsistency manager. These basic methods shall be obtained building on ideas of existing techniques for specific formalisms. Corresponding algorithms shall be developed by reduction to computational logic, in particular HEX programming might be exploited, akin to the mapping of MCSs in HEX-programs presented in Section 5.2. Again, optimizations w.r.t. scalability and efficiency are deemed to be crucial, and shall be achieved by semantic relaxations and/or syntactic restrictions.

7 Conclusion

We have briefly reviewed some nonmonotonic formalisms that allow to access external information sources, focusing on HEX-programs, recent modular logic programs, and multi-context systems, which have been developed at the Knowledge-Based Systems Group of the Vienna University of Technology in joint work with other researchers. In a systematic view, we have classified them according two distinguishing properties, namely the kind of environment view (local-model versus global-state) and the reduct (GL-style or FLP) used for the definition of the semantics; accordingly, the formalisms have different properties.

We have also compared the formalisms at a more fine grained level, pointing out similar behaviors on fragments and possible mappings between them (in particular, from MCSs to HEX-programs). Ongoing work is concerned with further developing the formalisms, and with applications based on them in research projects, targeting inconsistency management in heterogenous knowledge bases and query answering in distributed global knowledge bases.

Despite the progress in the last years, much more research efforts are needed in order to satisfy the growing need for formalisms with external information access, besides formalisms which are based on Answer Set Programming. Suitable semantics for collections of knowledge bases, pooled together in different settings will be needed (e.g., in small closed systems of a few nodes, and in open peer to peer systems where (many) nodes may dynamically enter and leave a system), which take peculiarities and pragmatic constraints into account (like network topology, communication cost, loss of messages etc.).

Developing efficient algorithms for reasoning in a distributed environment is a further challenging issue, in particular in the presence of nonmonotonic negation. For this, sophisticated optimization techniques are needed to increase the performance of simple prototype implementations satisfactorily. In the end, reasonable scalability for expressive formalisms still needs to be achieved. Nevertheless, we are confident that a success similar to the one recently seen in paradigms like SAT solving, CSP and ASP is possible in this area as well.

References

1. Analyti, A., Antoniou, G., Damásio, C.V.: A principled framework for modular web rule bases and its semantics. In: Proc. 11th Int'l. Conf. Principles of Knowledge Representation and Reasoning (KR 2008), pp. 390–400. AAAI Press, Menlo Park (2008)
2. Brewka, G., Eiter, T.: Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems. In: AAAI 2007, pp. 385–390. AAAI Press, Menlo Park (2007)
3. Brewka, G., Eiter, T.: Argumentation context systems: A framework for abstract group argumentation. In: LPNMR 2009. LNCS. Springer, Heidelberg (2009)
4. Brewka, G., Roelofsen, F., Serafini, L.: Contextual Default Reasoning. In: IJCAI 2007, pp. 268–273 (2007)
5. Calimeri, F., Cozza, S., Ianni, G.: External sources of knowledge and value invention in logic programming. Ann. Math. Artif. Intell. 50(3-4), 333–361 (2007)
6. Calimeri, F., Ianni, G.: External sources of computation for answer set solvers. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 105–118. Springer, Heidelberg (2005)
7. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Modular Nonmonotonic Logic Programming Revisited. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 145–159. Springer, Heidelberg (2009)
8. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Relevance-driven evaluation of modular nonmonotonic logic programs. In: LPNMR 2009. LNCS. Springer, Heidelberg (to appear, 2009)
9. de la Banda, M.G., Pontelli, E. (eds.): Logic Programming (ICLP 2008). LNCS, vol. 5366. Springer, Heidelberg (2008)
10. Drabent, W., Eiter, T., Ianni, G., Krennwallner, T., Lukasiewicz, T., Małuszyński, J.: Hybrid reasoning with rules and ontologies. In: Bry, F., Małuszyński, J. (eds.) Semantic Techniques for the Web: The REWERSE perspective. LNCS, vol. 5500, p. 50. Springer, Heidelberg (2009)
11. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77(2), 321–358 (1995)
12. Eiter, T., Fink, M., Krennwallner, T.: Decomposition of Declarative Knowledge Bases with External Functions. In: IJCAI 2009. AAAI Press, Menlo Park (2009)
13. Eiter, T., Gottlob, G., Veith, H.: Modular Logic Programming and Generalized Quantifiers. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS, vol. 1265, pp. 290–309. Springer, Heidelberg (1997)
14. Eiter, T., Ianni, G., Krennwallner, T., Polleres, A.: Rules and Ontologies for the Semantic Web. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2008. LNCS, vol. 5224, pp. 1–53. Springer, Heidelberg (2008)
15. Eiter, T., Ianni, G., Krennwallner, T., Polleres, A.: Answer set programming: A primer. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.-C., Schmidt, R.A. (eds.) Reasoning Web 2009. LNCS, vol. 5689, pp. 40–110. Springer, Heidelberg (2009)
16. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artif. Intell. 172(12-13), 1495–1539 (2008)
17. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In: IJCAI 2005, pp. 90–96. Professional Book Center (2005)
18. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective Integration of Declarative Rules with external Evaluations for Semantic Web Reasoning. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 273–287. Springer, Heidelberg (2006)

19. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining Answer Set Programming with Description Logics for the Semantic Web. In: KR 2004, pp. 141–151. Morgan Kaufmann, San Francisco (2004)
20. Eiter, T., Subrahmanian, V., Pick, G.: Heterogeneous Active Agents, I: Semantics. Artificial Intelligence 108(1-2), 179–255 (1999)
21. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 200–212. Springer, Heidelberg (2004)
22. Gelfond, M.: Answer sets. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, Foundations of Artificial Intelligence, ch. 7, pp. 285–316. Elsevier, Amsterdam (2007)
23. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: ICLP 1988, pp. 1070–1080. MIT Press, Cambridge (1988)
24. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and deductive databases. New Generation Computing 9, 365–385 (1991)
25. Giunchiglia, F.: Contextual reasoning. Epistemologia XVI, 345–364 (1993)
26. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics, or: How we can do without modal logics. Artificial Intelligence 65(1), 29–70 (1994)
27. Heymans, S., Toma, I.: Ranking services using fuzzy hex-programs. In: Calvanese, D., Lausen, G. (eds.) RR 2008. LNCS, vol. 5341, pp. 181–196. Springer, Heidelberg (2008)
28. Hoehndorf, R., Loebe, F., Kelso, J., Herre, H.: Representing default knowledge in biomedical ontologies: Application to the integration of anatomy and phenotype ontologies. BMC Bioinformatics 8(1), 377 (2007)
29. Lifschitz, V.: Twelve definitions of a stable model. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 37–51. Springer, Heidelberg (2008)
30. Lifschitz, V., Turner, H.: Splitting a Logic Program. In: ICLP 1994, pp. 23–37. MIT Press, Cambridge (1994)
31. Marek, V., Truszczyński, M.: Nonmonotonic Logics – Context-Dependent Reasoning. Springer, Heidelberg (1993)
32. McCarthy, J.: Generality in artificial intelligence. Commun. ACM 30(12), 1029–1035 (1987)
33. Nieuwenborgh, D.V., Cock, M.D., Vermeir, D.: Computing Fuzzy Answer Sets Using dlvhex. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 449–450. Springer, Heidelberg (2007)
34. Nieuwenborgh, D.V., Eiter, T., Vermeir, D.: Conditional Planning with External Functions. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 214–227. Springer, Heidelberg (2007)
35. Polleres, A., Feier, C., Harth, A.: Rules with Contextually Scoped Negation. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 332–347. Springer, Heidelberg (2006)
36. Roelofsen, F., Serafini, L.: Minimal and absent information in contexts. In: Proc. IJCAI 2005 (2005)
37. Ross, K.A.: Modular stratification and magic sets for datalog programs with negation. J. ACM 41(6), 1216–1266 (1994)
38. Schindlauer, R.: Answer-Set Programming for the Semantic Web. PhD thesis, Vienna University of Technology, Austria (December 2006)
39. Subrahmanian, V., Bonatti, P., Dix, J., Eiter, T., Kraus, S., Ozcan, F., Ross, R.: Heterogeneous Agent Systems: Theory and Implementation. MIT Press, Cambridge (2000)
40. Väänänen, J.: Generalized quantifiers, an introduction. In: Väänänen, J. (ed.) ESSLLI 1997. LNCS, vol. 1754, pp. 1–17. Springer, Heidelberg (2000)
41. Wang, K., Billington, D., Blee, J., Antoniou, G.: Combining description logic and defeasible logic for the Semantic Web. In: Antoniou, G., Boley, H. (eds.) RuleML 2004. LNCS, vol. 3323, pp. 170–181. Springer, Heidelberg (2004)