INFSYS

RESEARCH

REPORT

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

USING METHODS OF DECLARATIVE LOGIC PROGRAMMING FOR INTELLIGENT INFORMATION AGENTS

Thomas EITER Giuliana SABBATINI

Michael FINK Hans TOMPITS

INFSYS RESEARCH REPORT 1843-00-05

OCTOBER 2000

Institut für Informationssysteme

Abtg. Wissensbasierte Systeme

Technische Universität Wien

Favoritenstraße 9-11

A-1040 Wien, Austria

Tel: +43-1-58801-18405

Fax: +43-1-58801-18493

sek@kr.tuwien.ac.at

www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-00-05, OCTOBER 2000

USING METHODS OF DECLARATIVE LOGIC PROGRAMMING FOR INTELLIGENT INFORMATION AGENTS

Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits [1]

Abstract. At present, the search for specific information on the World Wide Web is faced with several problems, which arise on the one hand from the vast number of information sources available, and on the other hand from their intrinsic heterogeneity, since standards are missing. A promising approach for solving the complex problems emerging in this context is the use of multi-agent systems of information agents, which cooperatively solve advanced information-retrieval problems. This requests for providing advanced capabilities, such as search and assessment of information sources, query planning, information merging and fusion, dealing with incomplete information, and handling of inconsistency.

In this paper, our interest is in the role which some methods from the field of declarative (i.e., non-Prolog) logic programming can play in the realization of reasoning capabilities for information agents. In particular, we are interested to see in how they can be used, extended, and further developed for the needs of this domain of application. We review some existing systems and current projects, which typically address information-integration problems. We then focus on declarative knowledge-representation methods, and review and evaluate approaches and methods from logic programming and nonmonotonic reasoning for information agents. We discuss advantages and drawbacks, and point out the possible extensions and open issues.

[1] Institut für Informationssysteme, Abteilung Wissensbasierte Systeme 184/3, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria. E-mail: [eiter, michael, giuliana, tompits]@kr.tuwien.ac.at

Acknowledgements: This work was supported by the Austrian Science Fund (FWF) projects P13871-INF and N Z29-INF.

Contents

1 Introduction

To date, a user who is looking for some desired information on the World Wide Web faces several problems, which arise from the vast amount of available information sources and from their heterogeneity, since standards are missing. First of all, the user has to identify the relevant sources which should be queried, since bounds on resources, time, and/or cost of services usually do not permit to query *all* sources which are available. The generation of a query plan is often quite expensive and cannot be optimal when the user has no additional information about the knowledge contained in each source, but merely a short description of it. Then, once the user has made up his or her plan, for each source the query must be formulated in the appropriate way, depending on the interfaces available, as well as on the data organization and presentation. Furthermore, the user must possibly adapt the query more than once for each source in order to get the desired information, and must learn several different query approaches. Furthermore, it may happen that some sources provide as a result only partial or incomplete information to a query. The user has then to merge all data retrieved, taking into account that the information provided by different sources may be inconsistent. In such a situation, the user needs some notion of reliability for the sources in order to choose the proper information.

A promising approach for solving the complex problem outlined above is the use of a multi-agent system for accessing several heterogeneous information sources. The user is presented a uniform interface for accessing all available services and information sources, without having to bother with the heterogeneity underneath. It is the system as a whole which takes care of searching the appropriate sources, accessing them, and returning to the user the required information, and this to an extent as complete and consistent as possible.

The realization of such systems requests for special functionalities and capabilities, which emerge from specialized tasks like the ones outlined above. Such capabilities are provided as services on request by various kinds of information agents, which form a society of agents for cooperatively solving complex information-retrieval problems.

In this paper, we discuss some problems arising for the successful realization of information agents in a multi-agent environment, namely:

- search and assessment of information sources;
- query planning;
- information merging and fusion;
- dealing with incomplete information; and
- handling of inconsistency.

It is not hard to imagine that an advanced approach to any of these problems must involve, in some form, logical reasoning tasks, based on ad hoc knowledge about the task in question and on background knowledge of the domain, suitably represented in a knowledge base.

A number of different models and methods for knowledge representation and reasoning have been developed in the past, which may be used for this purpose. In this paper, our interest is in the role which some methods from the field of declarative (i.e., non-Prolog) logic programming can play in the realization of reasoning capabilities for information agents. In particular, we are interested to see how they can be used, extended, and further developed for the needs of this domain of application. As we shall see, the methods which are available to date do not meet all the needs which are required, which leaves the way for interesting research.

The paper is organized as follows. We first give a more detailed description of the role of information agents in the field of advanced information access, depicting some system architectures in Section 2. We then identify in Section 3 a set of reasoning capabilities which are mandatory for an advanced information-integration system, some of which are critical for an "intelligent" information agent. This will be followed by Section 4 with a short review of some existing systems and current projects which address typical information integration problems, most of them by means of ad hoc procedural techniques. Though we are more interested in declarative methods, the different approaches implemented in these systems are worth being analyzed, as they offer a good set of possible solutions together with some hints to intrinsic difficulties.

We then focus on declarative knowledge representation methods. First of all, Section 5 discusses some subtasks amenable for information agents, which promise to have a successful solution based on declarative methods. In the following Sections 6–9, we then review some approaches and methodologies in the field of declarative logic programming and nonmonotonic reasoning, which are likely to be of particular interest for one or more of the subtasks. In Section 10, an evaluation of the methods we examined follows, taking into account their applicability in the field of information agents. We discuss advantages and drawbacks, and address the possibility of extensions.

The final Section 11 presents some open issues which emerge from this analysis, and outlines directions for future work.

2 Intelligent Information Agents

Several models of multi-agent system have been developed during the last years, together with a number of successful applications, mainly in the fields of heterogeneous information and software integration, electronic commerce and Web interfaces. In this paper, we direct our attention on *knowledge-based approaches* to developing reasoning components for intelligent information access (for an extensive overview of agent programming approaches, see [132]). In particular, we focus on *information agents* (sometimes also called *middle agents* [50]) which are involved in the following tasks:

- finding, selecting, and querying relevant sources;
- managing and possibly processing the retrieved information; and
- updating their own knowledge about other agent services and features.

In this context, different scenarios—which to some extent overlap—have been suggested and implemented. Following [67], these can be classified as follows:

Facilitators: Agents which take control over a set of subordinated agents and coordinate the services they offer and the use of the resources they require.

Brokers: Agents often used for matching between a set of different data sources and user requests. Brokers receive requests, look for relevant sources matching them, and then perform actions using services from other agents (combining them with their own resources or information).

Mediators: In the mediator approach [140], meta-knowledge about a number of other agents (sometimes called *provider agents*) is available to the mediator, which exploits it to create higher-level services (not provided by the underlying agents) for user applications. These new services result by the combination and merging of low-level services on the basis of the comprehensive meta-information which the mediator has. In a sense, mediators may be seen as enhanced, high-level brokers.

Yellow Pages: A yellow pages dictionary helps users and other agents in finding the right providers (other agents or information sources) for the kind of service they need, possibly performing a match between advertised services, ontology of the domain in question and service requests.

Blackboards: A blackboard serves as a temporal repository for service requests which remain to be processed. Agents offering some service can access the blackboard and look for (and retrieve from it) service requests which they can satisfy.

Note that in a multi-agent system several of these agents with different features may be available at the same time, performing different tasks.

We are particularly interested in information agents which act as mediators. In that, we adopt the indispensable prerequisite that the agent has knowledge about the application domain of interest, and some meta-knowledge about the contents and features of the distributed, heterogeneous information sources the system has access to. For example, in an application where bibliographic references should be found, the agent must have knowledge about this domain in some form, and further knowledge about the information sources which it might consult, such as their reliability and availability.

For the goal of satisfying a user request, an information agent may have to solve various subgoals:

- identifying and possibly ranking relevant information sources;
- retrieving the required information (or ask some appropriate provider agent for it);
- processing the information returned by the sources by combining, merging, and integrating them;
- optimizing the number of accessed sources, or the total cost and time required for the search, or the precision and completeness of the results.

In these tasks, the agent has to make decisions about what to do or how to proceed at certain points. An "intelligent" agent makes them by reasoning using the knowledge that it possesses about the application and the particular request to be answered. Rather than having the decision mechanism implicitly hard-coded into the agent's program, we shall assume that the agent's decision making is strictly modularized and attached to a *thinking component*. Such a thinking component can be realized in many ways, by using one of the numerous approaches that have been developed in the agent and AI literature. In the present paper, we shall restrict our attention to the use of declarative (non-Prolog) logic programming for this purpose.

As for an agent-based information system, different architectures can be envisaged, in which information agents have different capabilities and roles. In the present paper, we shall assume the following system architecture:

- we have a multi-agent system providing access to several heterogeneous information sources. In such a system, the existence of an *interface agent*, which collects user requests (typically from the Web), and formulates them in a language suitable for subsequent processing is assumed. The user may be a person accessing a graphical interface, or may be some software.

- For each information source a specific *provider agent* takes care of access to that source (or a single provider agent manages the accesses to all sources of a given type; this is not relevant at this point). It receives a request in a standard communication language—ensuring homogeneity and implementation transparency—and accesses the underlying source as needed for retrieving the desired information. This access can consist of a query to a relational database in a standard query language, of an extensive search over some files, of a retrieval from databases containing special data (e.g., geographical data related to a given address), of a more complex search over the content of a set of Web pages, and so on. The provider agent composes then the information thus obtained into an answer in the standard communication language.

Information sources and corresponding provider agents may of course not be part of the system itself, the only requirement is the possibility of communicating with them in a standard language.

In such a context, (sets of) information agents act between interface and provider agents, in order to determine which sources are to be accessed and in which order, based on the user query, and to generate the appropriate answer based on the single answers of each accessed source, maybe resolving conflicts or handling incomplete information. Meta-knowledge about available sources and the application domain, as well as advanced reasoning and planning capabilities, are fundamental for this purpose.

An example of a multi-agent environment into which such a model of an information agent could fit into is IMPACT, the Interactive Maryland Platform for Agents Collaborating Together [31], which we shall shortly describe in Section 4.6. Note, however, that for the purposes of this paper, the particular architecture of an agent-based information system is not a primary issue. Rather, we are dealing with certain *capabilities* that should be realized in the system somehow, and for us it is less important where exactly this capability resides.

3 Problems and Challenges

In a multi-agent system for intelligent information access, information agents are supposed to provide advanced capabilities which help in improving the quality of query results. The following is a list of such capabilities, which partially overlap:

- **Decompose a request** in its "atomic" parts on the basis of available knowledge about the information sources, and reformulate it according to the corresponding data structure. E.g., if the user looks for a restaurant and wants to see where it is located, the query can be decomposed in two parts: a first search for a restaurant satisfying certain criteria and a second one for a picture showing a map of the corresponding location. This decomposition task can be of varying complexity and can generate a set of possible decompositions, based on the information about the underlying sources and the domain of interest [101, 102, 103, 44, 84, 28, 29, 30, 17, 19, 73, 74, 128, 2, 3, 4, 5].

- **Integrate the query** with additional user information, if available. This could require asking first a *profiling agent* for the user profile, habits, rights and interests. E.g., if a user is looking for Web pages about a given topic, and the system knows that the user does not speak English, then those pages containing text in his/her own language are to be preferred. This is useful not only for human users but for servicing other software as well. E.g., it is important to know if the user software poses requirements over the data format to be retrieved, if it has the possibility of directly communicating with the selected information source, if it has specific rights, and so on.

- **Select the information sources** to be queried, using the meta-knowledge about them and about the application domain to determine which of the sources contain relevant data to answer the query [101, 102, 103, 44, 84, 28, 29, 30, 4, 5, 17, 19, 73, 74, 128]. If possible, determine further preferred information sources to be queried.

- **Create a query plan.** Determine in which order sub-queries are to be performed [101, 102, 103, 17, 19], on the basis of query decomposition, of the data available in every source and of preference over sources, in order to optimize the execution. E.g., it could be required to minimize the costs or the expected waiting time for the user, to maximize the quality of the expected results, or to minimize the number of accessed sources and the network load, and so on.

- **Execute the plan**, asking the corresponding provider agents for the required services and waiting for their answers, possibly adapting the query plan dynamically to run-time information and to the answers obtained so far [101, 102, 17, 19, 94].

- **Compose and merge the answers.** This task can have varying complexity, from simply collecting all the answers and passing them to the interface agent without processing them, to organizing the retrieved data in some form (e.g., eliminating multiple instances of the same information, ranking the answers after their quality or after other criteria), to merging them in a single comprehensive answer, and so on [44, 84, 28, 29, 4, 5, 73, 74, 128].

- **Detect and possibly remove inconsistencies** among retrieved data [44, 84, 4, 5] on the basis of an inconsistency removal strategy and of meta-knowledge about the sources, or meta-knowledge contained in the answers themselves. E.g., some source may return data together with a reliability or probability value for those data.

- **Integrate incomplete information** by means of internal reasoning capabilities [73, 74, 128]. E.g., if in the geographical database there is no information available about a certain address, but the address is located in a certain district or near a known place, it could be useful to look for and show the map relative to the corresponding district, giving the user at least an approximative information.

- **Start a learning procedure** in order to improve or update internal models and reasoning rules [17, 19], perhaps providing input to the profiling agent as well. E.g., if a certain source is classified as providing an answer in one second, and it required for each of the last ten queries every time more than ten seconds, than the classification or the reliability of that source should be updated.

4 Systems and Frameworks

Some of the tasks for information agents identified in the previous section have already been widely addressed and some feasible solutions have been developed. Following the grouping suggested in [105], existing intelligent systems (in particular Web systems) have their main focus in some of the following fields:

- **User modeling and profiling:** addressing the issue of deploying adaptive user interfaces and recommendation systems, integrating user queries on the basis of the user profile or directly suggesting the user items of interest. We do not address this topic explicitly in our review; hints and relevant literature can be found in [105].

- **Analysis and preprocessing of information sources:** building up the meta-knowledge on which reasoning and decision making is based, on the basis of application domain description. As far as information agents are concerned, we will suppose that such meta-knowledge is available.

- **Information integration and information management:** covering a wide area of applications and different problems. We will focus on this global issue from the point of view given in Section 2.

In the following, we give an overview of some existing systems and frameworks as well as of current projects, which address different topics in the field of information integration. We briefly describe the problems they face, their approach and focus of interest, and their key features. Note that many of these systems use ad hoc procedural techniques, rather than declarative methods in which we are interested in. Nonetheless, they are highly relevant, since the kind of problems attacked by them and suggestions for solutions are of interest in the whole field of heterogeneous information integration. In the following sections, we consider how declarative formalisms (mostly based on declarative logic programming) can help in improving some rational capabilities of information agents. For a comprehensive overview of existing systems in the field of information integration and of the role of computational logic in some of them, see [56].

The common scenario of such systems is usually as follows. There is some intermediary software, which performs the tasks of provider agents as described in Section 2. The system can access several data sources and has some means of translating data in the required format for internal processing, as well as of translating requests in the format required by the information source. Users access the system and the available information sources through an interface which, like the interface agent we described in Section 2, hides from the user most of the implementation aspects and data representation issues. The user can query the system based on the semantics of the information he or she is looking for in a given application domain and receives from the system the required information, without having to care about different data representation schemas, formats, and languages.

4.1 Cohen's Information System for Structured Collections of Text

The system of Cohen [48] processes movie review data extracted from Web information sources. Text sources are converted into a highly structured collection of small fragments of text, on which automatic data processing is facilitated. Queries over this structured collection are then answered using an ad hoc "logic" called WHIRL [49], which combines inference with methods for ranked retrieval. The approximation is obtained using similarity metrics for text from information retrieval.

The system uses a relational data model in which every element of every tuple is assumed to consist of free text, with no additional structure. As two descriptions of the same object will not necessarily be identical, approximation of database operations with a similarity metric is performed. The output tuples are then sorted in decreasing order according to the similarity of corresponding fields. The method is proven to be robust to errors, e.g., in case of some field being incomplete due to extraction errors.

After off-line data collection, consisting of download and conversion to the required format, the following information is available for each document: a document vector representation, a human-readable summary consisting of the first 100 characters of the document and a reference to the original site of the document (URL). WHIRL returns for each query an HTML table which contains the best scoring answers.

The main advantage of this approach is that it requires little preprocessing work with respect to other similar tools, and no key normalization and no complete extraction of the documents are necessary.

An inference algorithm is used to optimize the search, combined with A* search methods and other methods of conventional information retrieval systems.

4.2 Information Manifold

The Information Manifold [101, 102, 103] provides uniform access to autonomous, heterogeneous, structured information sources on the Web, which make use of different data models and vocabularies.

The language CARIN, due to Levy and Rousset [104], based on description logics, is used to describe the structure and properties of existing information sources and to add new ones. The query answering algorithm guarantees that only relevant sources are accessed, while further pruning on the basis of run-time information is performed. Queries are expressed by means of classes and roles in CARIN, ordinary predicates of every arity, and some specific predicates.

The user is presented with a world view consisting of a set of relations with a class hierarchy, modeling the available information about the domain of interest. Presented relations are only virtual: the actual data remain stored in the external sources with their own structure. Only the descriptions are centralized and re-organized. Queries can be formulated in terms of the world view relations, and the system uses the available descriptions of the sources to generate a query-answering plan (see [101, 102]). The aim of the planning tool is the minimization of the number of information sources to be accessed.

4.3 Carnot

Carnot [84, 44] is a system of federated heterogeneous databases based on static data integration, and constitutes the foundation for the InfoSleuth project presented in the next subsection.

Many heterogeneous information models (i.e., schemas of databases, knowledge bases and process models) are statically integrated by means of an ad hoc method, achieving integration at the semantic level by using an existing global ontology of the application domain to resolve possible inconsistencies.

An existing global schema is combined with the schemas of the individual sources, comparing and merging them with the global one but not with each other. Axioms are then produced stating the equivalence between some components of two theories, capturing the mapping between the data structure of each individual source and the global schema. In this way, a global view of all information sources—and at the same time local views corresponding to each source—are available.

Carnot is reported to have been successfully applied in the fields of scientific decision support over several large scientific databases, workflow automation and management from a database perspective for telecommunication industry [85], data validation and cleaning of large databases, information retrieval from structured databases in response to text queries through model integration of heterogeneous resources.

4.4 InfoSleuth

InfoSleuth [28, 29, 30] is a Web agent system based on the results of the Carnot project. It uses suitable domain ontologies which provide a declarative description of semantic information inherent to a specific domain, independently of possibly different system representation. A semantic matching between requests and sources is then performed, based on the corresponding domain ontology.

Information sources are thus considered at the level of their relevant semantic concepts. Requests can be generically specified independent of structure, location, and existence of data, and are then matched with the relevant information sources at the time of processing. This allows the matching procedure taking into account also run-time information, e.g. the availability of a Web database.

Traditional querying and schema mappings are used together with the KIF (*Knowledge Interchange Format*) knowledge representation language [80], which enables programs with different internal knowledge representation schemas to exchange information in a common language. The internal format is converted to KIF and then wrapped in KQML (for *Knowledge Query and Manipulation Language*) performatives.

Queries are routed by appropriate mediators and brokers to data retrieval agents. Each agent can process the requests it receives by making inferences on local knowledge in order to answer some part of the query, or by routing the request to other agents, or by decomposing it into sub-requests. Decisions are based on the domain ontology and on the meta-data describing agent knowledge and relationships with other agents.

InfoSleuth has been applied in the field of health care over distributed heterogeneous databases of hospitals for comparing results of the treatment, and in the environmental domain as a component of *EDEN* (Environmental Data Exchange Network).

The approach has some drawbacks and weaknesses, though [28]:

- ontologies must be sufficiently general to ensure that potentially new information maps to the same ontology; at the same time, exact mappings have to be possible;
- the semantics of ontological concepts may be incomplete, uncertain, and evolving over time, therefore the system should be able to handle these dynamical aspects;
- retrieval and extraction from text resources should be improved;
- different levels of information aggregation are needed, requiring complex hierarchies in the ontology;
- credibility and reliability of information sources and of single information items have to be considered, which raises the need for additional meta-data;
- aggregation accountability should be implemented in order to improve the services available for the user, requiring for traceability of the combinations;
- above all, probability and membership functions for combining and propagating probability measures through aggregations are needed, complex ontology hierarchies with many degrees of similarity between concepts should be handled.

4.5 HERMES

The Heterogeneous Reasoning and Mediator System (HERMES) [4, 5, 6, 136] is based on a general declarative language for the definition of a mediator [140], which expresses semantic integration of information from diverse data sources and reasoning systems. A facility enables the incremental integration of new systems into an already existing mediator.

The system addresses two main issues:

1. domain integration, i.e. the physical linking of the available data sources and reasoning systems; and
2. semantic integration, i.e., the coherent extraction and combination of the provided information.

Methods to detect potential conflicts and to resolve them, to pool information together, and to define new compositional operations over information can be specified and stored.

External data sources and software packages are accessed by means of the API they provide, such that HERMES has to manage a translation of user queries into the required syntax for these APIs, and to convert the delivered answers into a suitable format for further processing.

In a specific mediator programming environment, a suitable mediator compiler and some yellow page servers are made available through a user interface.

Applications have been developed for the integration of terrain maps on the basis of relational databases, as well as for face recognition in law enforcement.

4.6 IMPACT

In the Interactive Maryland Platform for Agents Collaborating Together (IMPACT) [20, 31, 62, 63], several heterogeneous agents can interact with each other and with some IMPACT servers by means of *wrappers*, one for each piece of software serving as agent. IMPACT servers provide some infrastructural services the agents need in order to cooperate, including yellow pages, thesaurus, and registration.

The decision-making component of each agent is declarative and comprises several items, among them an agent program. This program contains rules which govern the agent's behavior. They are based on a logical abstraction of the internal functions and data structures managed by the agent, together with the current state of knowledge. Agents can also hold beliefs about other agents and make their decisions depending on these beliefs, as well as on general constraints and action policies. A precise semantics for agent programs is provided, formally describing the behavior of agents.

External software can, by means of a logical abstraction, act and interact with other agents present in the platform: IMPACT offers in this way the possibility of integrating and reusing legacy software, making its services available in such a way that implementation aspects are not relevant for the interaction.

The system has been successfully used in the development of a logistic application for the US Army Logistic Integration Agency War Reserves planning. Other example applications have been developed for department stores, for solving the *controlled flight into terrain* problem, and for supply chain management.

4.7 SIMS

The Single Interface to Multiple Sources effort (SIMS) [17, 19] exploits the semantic model of a domain to integrate the information from various sources, making use of techniques for knowledge representation and modeling, for planning and search, as well as for learning.

The implemented system can be combined with Oracle databases as well as with LOOM [111] knowledge bases. Specific tools are used for generating database queries in the appropriate language and for query planning.

The main aim of reasoning tasks in the system is to determine which information sources are to be accessed, by matching and reformulating the queries, and then to create and optimize the corresponding query plan. This is achieved through a common domain model, consisting of objects and possible actions, which are then mapped to the single databases.

The approach has been applied to transportation planning problems for the movement of personnel and material from one location to another [18], to the control of the status of the U.S. Pacific Fleet [16], and to decision support systems for medical trauma care.

4.8 TSIMMIS

The Stanford-IBM Manager of Multiple Information Sources (TSIMMIS) [73, 74, 128] integrates heterogeneous information sources including structured and unstructured data, to be used in decision support systems.

The architecture of the system includes several components addressing different issues:

1. a **classifier/extractor** for the extraction of properties from unstructured objects; this component is based on the RUFUS System developed at the IBM Almaden Research Center;
2. a **translator/wrapper** for the translation of information into a common object model, also performing data and query conversion;
3. a **mediator** combining information from the several sources;
4. a **browser**.

A specific tagged object model and a corresponding query language over its objects, adapting existing SQL-like languages for object-oriented models, are used in the system [74].

LOREL [128, 3, 2] is the second flexible query language that has been developed by ad hoc methods and is used in TSIMMIS for handling semistructured data. The choice of developing a new language instead of using an already existing one was motivated by the necessity of covering some features which are not addressed by other languages. Queries should be

- answered meaningfully, even when some data is missing, by means of partial type/object assignment;
- handled uniformly over single- and set-valued attributes, to manage data with irregular structure;
- manipulated uniformly over data of different types, in order to deal with databases where different types refer to the same concept;
- possibly return heterogeneous sets;
- executable even if the object structure is not fully known, by means of wild-cards over attribute labels.

4.9 Further Projects

Apart from the projects described in the previous subsections, a number of other projects are in progress which explore alternative promising techniques in the field of information integration and management. We give here a short overview of some of them.

4.9.1 MENTAL

This project at Lisbon University, directed by L.M. Pereira, consists of four main tasks, which partially overlap:

Agent architecture, coordination and task integration, dealing with:

- integration of abduction and tabling mechanisms;
- expressing updates by means of abduction;
- constraints, strong negation and abduction.

The reference architecture [55] combines KS-Agents (by Kowalski and Sadri [96]), dynamic logic programming paradigm for updates (cf. Section 7.5), tabling and abduction, and it is built upon the XSB-Prolog system [129, 133].

A framework for communicating agents has been suggested, in which agents have the ability of dynamically updating their knowledge base and their goals as a result of interactions with other agents, based on forward and backward reasoning.

Knowledge management and reasoning, dealing with negation and paraconsistency, updating of logic programs [9, 10, 13], belief revision and abduction on the basis of abductive extended logic programs. A prototype application to alarm-correlation in cellular-phone networks has been implemented [71].

Actions, interactions and cooperation. Specialized sub-agents have been suggested taking care of different aspects in multi-agent systems:

- actions and observations [14];
- coordination of interaction protocols;
- individual and joint planning of actions;
- non-interference behavior;
- resource sharing;
- self and mutual diagnosis of imperfect knowledge.

Learning and strength of beliefs, addressing the following theoretical issues [99]:

- reinforcement learning;
- causal deductive back-propagation;
- genetic algorithms for individual and cross-agent learning applied to logic programming;
- degrees of belief based on the strength of supports of beliefs;
- truth maintenance based learning in logic programming.

4.9.2 Logic Programming for Multi-Agent Systems

The role of logic programming in multi-agent systems is quite always confined to the reasoning and decision components of agents, whereas other functionalities required in the traditional *observe-think-act* cycle of the agent are provided by means of other programming approaches. As Kowalski and Sadri suggest in [97], it is nevertheless possible to extend logic programming to provide other features required in multi-agent systems, with the purpose of unifying the rational and the reactive components of an agent in a single framework. From this point of view, the approach extends logic programming to handle multi-theories in a shared environment, in order to take into account the knowledge bases of multiple agents, and to model not only queries and updates in form of observations but also message exchange and actions in the system.

The BDI architecture of Rao and Georgeff [81] can also be seen as a special case of this framework, in which agent programs handle uniformly different concepts such as beliefs and goals, queries, obligations and prohibitions, integrity constraints, rules of the form *"if condition C holds, then execute action A"*, actions to be performed, and commands.

Interesting applications of this extension of logic programming deal with distributed information integration, in which agents act as mediators to perform query optimization over heterogeneous sources, and with information network management, with the purpose of checking and enforcing structural and content properties over the nodes in the network.

4.9.3 MIA: Mobile Information Agents

The goal of this project is to develop intelligent information systems which provide a mobile user with information extracted from the Web, related to the spatial position of the user. The location of the user is checked and updated through a GPS receiver, so that the system is able to answer user queries on a specific topic with information of local relevance. E.g., a user comes late in the night to some city and wants to know which restaurants are still open offering some particular kind of food.

Information search and extraction from the Web is performed by specific information agents ahead of the user requests. When the user (i.e. the client software) asks for some information, the server side provides it with the current knowledge present in the system, with respect to user location. The user is not presented with a list of links to Web pages containing relevant information, but directly with the required information, appropriately preprocessed and merged.

The agents building up the knowledge of the system are autonomous in detecting Web pages of interest to some specific topic and in extracting, based on their structure and content, relevant information to be incorporated in the knowledge of the system. The representation and reasoning components are based on logic programming and automated theorem proving together with machine learning techniques. For more details about the project, see [137, 138].

5 Declarative Methods

5.1 Overview

When looking for feasible or improved solutions to the main problems of advanced information access, which we recalled in Section 3, some central reasoning sub-tasks can be identified, in form of basic representation and reasoning capabilities which the information agent itself or some designated modules must have. For some of these reasoning tasks, listed below, declarative methods seem to be promising.

Priority handling. Dealing with priority in the logic programming context has received considerable attention, see e.g. [79, 35, 89, 68, 53, 34, 38, 39, 12]. Priority information needs to be encoded in the knowledge base of the agent, possibly explicitly in the object language in which the knowledge itself is expressed. The preference relation may be static or dynamic, in the latter case reasoning procedures have to account for possible changes.

Revision and update. The knowledge base of the agent can be subject to change on the basis of information from the environment (the application domain itself, other agents inside or outside the system, etc.), in order to incorporate changes in the outside world or changes in the knowledge about it [34, 112, 100, 88, 134, 90, 69, 70, 9, 10, 11, 12, 13]. The epistemic state and the intended belief set of the agent have thus to be continuously revised, depending on some specified update policy, which could be in turn explicitly described in the knowledge base and maybe dynamically updated too.

Inconsistency removal. The agent should in the presence of conflicts in the knowledge base be able to detect them and to find an acceptable *fall-back* knowledge configuration, in order to ensure that the decision making process is not stopped or inconsistent as a whole. The fall-back configuration is usually required to preserve "as much as possible" of the available knowledge.

Decision making with incomplete information. Under the manifestation of incomplete information, some reasoning and deduction capabilities may provide candidate hypotheses for the missing information pieces, in order to ensure that the process of decision making can derive plausible results.

Temporal reasoning. The evolution of a dynamic knowledge base could be subject to specifications [13], and corresponding forms of reasoning about this evolution could be provided for ensuring that the agent's behavior is appropriate, e.g., that some undesired status cannot be reached [64, 47].

Learning. Based on the history of the agent (sequence of changes in its knowledge base, or sequence of observations), some form of inductive learning could be implemented [134, 90, 55]. We do not further address this topic in the rest of this paper.

In the following sections, we will address declarative formalisms, mainly based on declarative logic programming, which offer partial or tentative solutions for the implementation of such capabilities. We shall point out in which way and for which purposes the individual approaches seem to be more appropriate.

Note that besides methods from logic programming, other knowledge representation and reasoning techniques could be suitable as well (e.g., description logics). However, to keep this review focused, we do not consider such methods in this review.

Furthermore, we do not focus here on applying particular modes of reasoning such as abduction or induction. We refer to [132, 66, 91] and references therein for further information.

5.2 Short Review of Logic Programs and Answer Sets

We conclude this introductory section with briefly recalling some basic concepts and notions of declarative logic programming, which shall be encountered in the following sections. In our review, we deliberatively do not focus on all technical details, however, and thus our descriptions of the various approaches will be superficial.

A logic program consists of a (finite) collection of rules

$$Head \leftarrow Body,$$

where $Head$ is a literal and $Body$ is a (possibly empty) conjunction of literals. Rules where $Body$ is empty are called *facts*. Literals are built over a set of atoms from a first-order language, using default negation not (negation as failure, also called weak negation), and strong negation $\neg$ (referred to as classical negation in some texts, e.g. in [77]). The application of $\neg$ is restricted to atoms; thus, if A is an atom, then $not\neg A$ is allowed, while $\neg not\ A$ is not allowed.

Example 1 *Consider the following rules, representing knowledge of an agent reasoning about the problem which of two information sites should be queried:*

$$\begin{array}{lrl}
r_1: & up(S) & \leftarrow not\neg up(S); \\
r_2: & \neg query(S) & \leftarrow \neg up(S); \\
r_3: & query(index) & \leftarrow not\neg query(index),\ up(index); \\
r_4: & query(cite_seer) & \leftarrow not\neg query(cite_seer),\ not\,up(index),\ up(cite_seer); \\
r_5: & flag_error & \leftarrow \neg up(index),\ \neg up(cite_seer).
\end{array}$$

Informally, rule r_1 says that a site is up by default, and r_2 that if a site is down, then it cannot be queried. The rule r_3 states that by default, the science citation index site ($index$) is queried if it is up, while rule r_4 states that by default the $cite\_seer$ site is queried if it is unknown whether $index$ is up but known that $cite\_seer$ is up. Rule r_5 says that if both $index$ and $cite\_seer$ are down, then an error is flagged. The rules r_2 and r_5 are strict, since they apply in all contexts, while the other rules are defeasible.

If, as in the above example, default negation does not occur in the heads of rules, then a program P is called an *extended logic program* (ELP). If, additionally, no strong negation occurs in P, i.e., the only form of negation is default negation in rule bodies, then P is called a *normal logic program* (NLP). The generalization of an NLP by allowing default negation in the heads of rules is called *generalized logic program* (GLP).

Further variants of logic programs exist. Some of them allow rules with empty heads (called *constraints*) and/or rules with a disjunction $L_1 \vee \cdots \vee L_n$ of literals in the head. In the latter case, programs are called *disjunctive logic programs* (DLPs).

The semantics of a logic program P is defined in terms of its ground instantiation $ground(P)$, which contains all ground instances of rules over the Herbrand universe that is generated by the function and constant symbols occurring in P. The ground instance $ground(P)$ may be viewed as a propositional program, where each ground atom is a propositional atom.

There are several semantics of ELPs [57], one of the most important is the concept of an answer set [77], discussed below, which generalize the concept of a stable model for NLPs [76]. Similar definitions for GLPs and other classes of programs can be found in the literature (see, e.g., [108]).

A (*partial*) *interpretation* is a set I of classical ground literals, i.e., literals without not, such that no opposite literals $A, \neg A$ belong to I simultaneously. A classical literal A (resp. $\neg A$) is true in I, if A (resp. $\neg A$) belongs to I, while a default literal $notA$ (resp., $not\,\neg A$) is true in I if A (resp., $\neg A$) does not belong to I. An interpretation I satisfies a ground rule r, written $I \models r$, if either the head of r is true in I, or some literal in the body of r is not true in I.

Given an interpretation I and an ELP P, the *Gelfond-Lifschitz reduct of P with respect to I*, denoted P^I, is the program obtained from $ground(P)$ as follows:

1. remove every rule r from $ground(P)$ which contains in the body a weakly negated literal that is not true in I;

2. remove all weakly negated literals from the remaining rules.

An interpretation I is an *answer set* of a program P if it coincides with the smallest set of classical literals which is closed under the rules of P^I (note that this definition excludes inconsistent answer sets, which were allowed in the original definition).

Example 2 *For illustration, let us reconsider the program P which contains the rules r_1–r_5 from Example 1. The set $I = \{up(index), up(cite\_seer), query(index)\}$ is an interpretation of P. Moreover, it is an answer set of P: The reduct P^I contains the rules*

$$\begin{aligned}
up(index) &\leftarrow ;\\
up(cite_seer) &\leftarrow ;\\
\neg query(index) &\leftarrow \neg up(index);\\
\neg query(cite_seer) &\leftarrow \neg up(cite_seer);\\
query(index) &\leftarrow up(index);\\
flag_error &\leftarrow \neg up(index), \neg up(cite_seer).
\end{aligned}$$

Each set of literals which is closed under the rules of P^I must contain the literals $up(index)$, $up(cite\_seer)$, and, by virtue of the penultimate rule of P^I, also $query(index)$. No further literals need to be added. Thus, I is closed, and therefore it is an answer set of P.

In general, an ELP might have multiple answer sets.

Example 3 *Suppose we add to the program P of Example 1 the rule*

$$r_6: \quad \neg query(S) \leftarrow not\, query(S), \neg rel(S)$$

and the facts $\neg up(index) \leftarrow$ and $\neg rel(cite\_seer) \leftarrow$. Intuitively, the rule r_6 states that an unreliable source is not queried by default. Then, the corresponding ELP Q has two answer sets:

$$\begin{aligned} J_1 &= \{\neg up(index), \neg rel(cite\_seer), up(cite\_seer), \neg query(index), query(cite\_seer)\}, \\ J_2 &= \{\neg up(index), \neg rel(cite\_seer), up(cite\_seer), \neg query(index), \neg query(cite\_seer)\}. \end{aligned}$$

They correspond to the fact that in this scenario, both deciding for and against querying $cite\_seer$ is acceptable.

The set of consequences of a logic program P is usually defined as the set of literals which are cautiously derived from it, i.e., which are true in all answer sets of P. Thus, in Example 1, $query(index)$ is a consequence of P, while neither $query(cite\_seer)$ nor $\neg query(cite\_seer)$ is a consequence of P.

If, as in the Example 3, multiple answer sets exist, preferences might be used to eliminate some of the answer sets to single out either a unique "preferred" answer set, or a collection of preferred answer sets. If, for instance, rule r_4 would have preference over the rule r_6, then J_1 would be the single preferred answer set of Q, and $query(cite\_seer)$ would be among the conclusions of Q. There are different ways to represent and handle preferences on logic programs, and we shall review some of them in the next section.

As for the issue of updates, suppose the fact $\neg up(index) \leftarrow$ becomes known and should be incorporated into P. If we simply add it, then the resulting program P' would have the answer set

$$I' = \{\neg up(index), up(cite\_seer), \neg query(index), query(cite\_seer)\}.$$

Thus, $query(index)$ is retracted while $query(cite\_seer)$ is added. Simply adding a new piece of information to a knowledge base given by a logic program is, in general, not a suitable strategy for updating the knowledge base. We discuss a number of approaches to updating logic programs in Section 7.

6 Preference Handling

An important requirement for an advanced information agent is the capability of representing and handling priority information. Such information may be either represented as meta-data associated with the rules in the knowledge base, which is used in the process of computing the intended (or preferred) belief set, or associated with single rules or whole programs when revising or updating the knowledge base itself.

As meta-data associated to the rules in the knowledge base, preference information can be useful to operate choices in case of multiple possibilities (e.g., if a logic program encoding an agent's decision component has more than one answer set), or to determine rules which should be overridden in case of conflicts (e.g., the logic program as a whole has no answer set, but retracting a rule with low priority makes it consistent).

When associated with (sets of) rules serving as updates of the knowledge base, preferences clearly express the priority of the new information with respect to the current knowledge. These preferences can come as meta-knowledge provided by other source agents (e.g., indicating the precision of the required information) or can be part of the meta-knowledge of the agent itself (e.g., knowledge about the credibility of a certain source).

Several declarative methods have been developed in the field of logic programming to cope with prioritized information, which differ in the following respects.

Meta- vs. object-level encoding. Priorities can be explicitly expressed in the object language of the knowledge base (Sections 6.1, 6.4, 6.5, 6.7), usually by means of a special symbol in the alphabet and of a naming function for rules; alternatively, they can be given as meta-knowledge associated to the rules (Sections 6.2, 6.4, 6.6, 8.1) or to the literals (Section 6.3), in form of a (partial) order among them.

Static vs. dynamic priorities. Priorities can be static or dynamic: static priorities are not explicitly contained in the knowledge base and are not subject to change (Sections 6.2-6.4, 6.6, 8.1); on the other hand when priorities are expressed in the object language (Sections 6.1, 6.4, 6.5, 6.7), the preference relation can itself be dynamically revised or updated together with the other knowledge.

Fixed vs. variable strategy. Strategies for priority handling can be fixed, usually described through an appropriate ad hoc semantics taking care of respecting the preference relation (Sections 6.2-6.4, 6.7, and 8.1), or they can be encoded in the knowledge base of the agent by means of appropriate program rewriting (Sections 6.5, 6.6) or by addition of appropriate rules (Section 6.1). In this case, no special semantics is usually needed, and the handling policy could itself be subject to change, or could easier be modified for other purposes.

6.1 Prioritized Defaults by Gelfond and Son

In [79], a method is suggested for reasoning with prioritized defaults in extended logic programs under the answer set semantics. To this purpose, a set of axioms, in forms of an ELP defining how prioritized defaults are to be handled, is introduced. This set of axioms, added to any logic program with priorities between the defaults, produces the desired results. No new semantics is needed, as the answer set semantics of the union of the given program and of the set of axioms works as intended.

In this approach, defaults and rules are distinguished, as rules are non defeasible, while defaults express usual properties of some elements. Dynamic priorities are addressed, since defaults and rules about the preference relation are permitted. Moreover, changes in the properties of the preference relation, in the definitions of conflicting defaults, or in the way of reasoning only require simple changes in the formalism. Alternative strategies for priority handling can thus be implemented by necessary changes in the set of axioms.

6.2 Preferred Answer Sets by Brewka and Eiter

In [35], two general principles for prioritized knowledge representation formalisms are formulated and analyzed, and a notion of static preferences over rules is introduced, satisfying these principles and extending the answer set semantics for ELPs (an extension to dynamic preferences was given in [36]).

Based on a given strict partial order over rules, some of the answer sets of the ELP are selected as "preferred", therefore increasing the set of conclusions which can be cautiously derived from it.

A *strong* and a *weak* notion of preferred answer sets is considered, respectively, whereas the second one ensures that if a program has an answer set, then it has also at least a preferred one, which is not guaranteed with the strong definition, i.e., a program may have answer sets but no preferred one.

The basic idea behind this approach is that the order in which rules are applied has to be compatible with the explicitly given preference among rules. A rule can only be applied in case its body is not defeated by another rule having higher priority.

6.3 Prioritized Logic Programs by Inoue and Sakama

In [89], the problem of reasoning with explicit representation of priorities in logic programming is addressed. The suggested approach serves mainly to reduce nondeterminism in choosing one out of several answer sets of a given program, and can capture several forms of reasoning, e.g., abduction, default reasoning, and prioritized circumscription.

The semantics of the so called prioritized logic programs (referred to as PLPs) is given by means of appropriate preferred answer sets. PLPs are constructed as general extended disjunctive programs (GEDPs, including normal programs and able to express abductive programs as well), together with a reflexive and transitive priority relation over literals of the language.

The semantics of a PLP is defined in terms of the answer sets of the corresponding GEDP, through an appropriate notion of preferred answer sets. Intuitively, from the notion of priority over the elements of the language, priorities over answer sets are derived. In case the set of priority relations is empty, the given definitions reduce to the standard semantics of answer sets.

Priorities over the language can also be used to express more general priorities, introducing new elements as heads of suitable rules and priorities over these new elements.

In [90], abduction is used to derive appropriate priorities yielding to intended conclusions. In particular, an appropriate set of priorities is derived which can explain an observation under skeptical inference, thus addressing the multiple extension problem of answer set semantics for ELPs.

To this purpose, the notion of preference abduction is introduced, in order to infer a sufficient priority relation to make the intended conclusion hold. This is achieved through an integrated framework for the abduction of both literals and priorities. The basis is that of prioritized and abductive ELPs.

In a first step, a basic framework is introduced. This step selects, through abduction, from a given set of candidate priorities, those which are sufficient to derive an observation skeptically, when the given observation is credulously but not skeptically derived by the initial program.

An extended framework is then obtained by combining preference abduction with ordinary abduction, when the given observation cannot even credulously be inferred by the initial program. In this case, given a set of abducibles and a set of priority relations, subsets of abducibles and of priorities are selected such that the initial program together with new literals and with the given priorities, is sufficient to explain the observation skeptically.

In case the set of abducibles is empty, the extended framework collapses to the basic one. In case the set of candidate priorities is empty, the framework collapses to abductive logic programming. If the set of abducibles is made up of rules instead of literals, and the set of priorities contains relations over those rules, a naming technique can be used to yield the defined framework. Moreover, this framework can also be used to derive possible new abducibles starting from the rules which are the source of the nondeterminism. These are in turn converted to abducible rules and then renamed, in order to come to new abducibles. Preferences over abductive hypotheses can also be used to acquire new priority information.

6.4 Prioritized Logic Programs by Foo and Zhang

In [68], PLPs with static priorities under an extended answer set semantics are introduced. A PLP consists in this approach of an ELP, a naming function for rules, and a strict partial order on names. The idea is to reduce PLPs to ELPs by progressively deleting rules that, due to the defined priority relation, are to be ignored. A rule is ignored if a rule of higher priority exists such that keeping the latter and deleting the former still results in defeating the rule with lower priority. If this is not the case, even if one rule is more preferred over the other, the two rules do not affect each other during the evaluation, i.e., the preference relation between these two rules does not play any role. Therefore, by progressively checking all the rules with respect to the given order, the rules that are going to be ignored can be deleted, and for the remaining ones the preference order plays no role at all. This can be carried out until a fixed point is reached, in form of an ELP with no priority relation. The answer sets of this program are the intended answer sets of the initial PLP.

In [68], PLPs are extended to express dynamic priorities. The semantics of a dynamic PLP is defined as the answer set semantics of the corresponding (static) PLP.

A dynamic PLP is a pair consisting of a program over a language containing a symbol expressing a partial ordering between names of rules, and a naming function for rules. In order to ensure the consistency of the given partial ordering, the presence of some special rules in the program (expressing antisymmetry and transitivity of the preference relation) is assumed.

The semantics is given in terms of a transformation from dynamic PLPs into (static) PLPs, through a sequence of reductions with respect to the ordering. Starting from a dynamic PLP, successive PLPs are built up until a fixed point is reached. Each new obtained program is a reduct of the previous one with respect to the order relation over the previous program, and the new order relation contains all pairs such that the corresponding rule belongs to all answer sets of the previous PLP. The answer sets of the dynamic program are the answer sets of the result of this transformation.

6.5 Compiled Preferences by Delgrande et al.

In [53], a methodology is suggested to compute the answer sets of ELPs with dynamic priorities between rules, explicitly expressed in the program itself (such a program is called *ordered*). The ordered program is transformed, on the basis of the encoded preferences, into another ELP, whose answer sets correspond to the intended preferred answer sets of the original one. The central idea of the approach is the introduction of special purpose atoms together with a transformation of the original program in a set of rules over these new atoms, ensuring the desired handling of preferences. The methodology is shown to be powerful enough to capture other formalisms as well, in which preferences are statically expressed outside the program, in the form of meta-information. Moreover this formalism can be extended to deal with preferences among sets of rules.

Alternative strategies for preference handling can be achieved as well by changing the specifications of the transformed program. For instance, in [52] it is shown that the method of Brewka and Eiter can be encoded within this framework.

6.6 Disjunctive Programs with Inheritance by Buccafurri et al.

In [39, 38], a framework for disjunctive logic programs with inheritance (denoted $DLP^{<}$) on the basis of answer set semantics is introduced. Given a hierarchy of objects represented by disjunctive programs, possible conflicts in determining the properties of each object in the hierarchy are solved by favoring rules which are more specific according to the hierarchy. The hierarchy is given in terms of a strict partial order over objects. If, for simplicity, we identify each object with an indexed program, a hierarchical knowledge base consists of a set of programs and a partial order on them. The program for a given object in the hierarchy is thus given by the collection of programs at and above the corresponding indexed program. The semantics of inheritance programs is then defined as an extension of answer set semantics.

6.7 Preference Default Theories by Brewka

In [34], the problem of handling preference information in epistemic states is addressed. Preference information is considered as explicitly expressed in the representation language of epistemic states, seen as nonmonotonic theories.

The desired results are achieved in two steps. The first step consists in an extension of default systems in order to express preference information in the object language, together with an appropriate new definition of theory extensions. In a second step, a notion of prioritized inference is introduced, as least fixed-point of an appropriate monotone operator, thus identifying epistemic states with preferential default theories under this ad hoc semantics. It has to be pointed out that, due to potential self-referentiability of preference information, not all theories expressed in the new formalism possess extensions (i.e., acceptable sets of beliefs).

The differences between the new formalism and the original default notions consist in the following features:

- a single set is considered containing all information, in which there is no absolutely unrevisable information;
- preference and other meta-information are expressed in the object language, by means of a naming mechanism for formulas and a strict total order between names.

A new notion of extension for these preference default theories is given: an extension is generated by a strict total order on the (finite) theory itself, building up consistent subsets of the theory with respect to this total order.

A preferred extension is the deductive closure of a maximal consistent subset of the theory, which can be generated through a total preference ordering on the formulas of the theory, compatible with the preference information explicitly expressed in the formulas belonging to the extension.

A formula is an accepted conclusion of a preference default theory if it belongs to the least fixed-point of the operator computing the intersection of all preferred extensions of the theory.

The idea is to iteratively compute, starting with the empty set, the intersection of the extensions which are compatible with the information obtained so far. Since the set of formulas in each step may contain new preference information, the number of compatible extensions may be progressively reduced, and their intersection may thus grow. This process is continued until a fixed-point is reached. If a formula is an accepted conclusion of the theory, then it is contained in all preferred extensions of the theory. The set of accepted conclusions of a theory is consistent and, by identifying the theory with an epistemic state, the set of accepted conclusions defines the corresponding belief set.

In Section 7.8, we discuss the use of this formalism to represent revision strategies.

6.8 Further Approaches

Besides the approaches that we have discussed in the previous subsections, a number of other approaches for adding priorities in extended logic programs are available. We briefly address some of them in this section.

Under the answer set semantics, ELPs may be viewed as a fragment of Reiter's default logic, such that the answer sets of an ELP P correspond one-to-one to the consistent extensions of of the corresponding default theory [77]. A semantics for prioritized ELPs is thus inherited from any semantics for prioritized default logic, such as the proposals discussed in [114, 32, 21, 131]. Applied to ELPs, all these semantics select, by using priority information on rules, particular answer sets of an ELP from the collection of all answer sets. They fail, however, to satisfy the principles for this selection process which have been proposed in [35].

Further semantics for priorities in extended logic programming, which are not based on answer sets, have been proposed in [122, 15, 33, 126]. They have quite different foundations such as logical entrenchment and specificity [122], reliability handling [15], well-founded semantics [33], or defeasible argumentation [126].

In [125], it is shown how priorities can be used to combine different potentially conflicting Datalog databases. Preferences are used to determine information which is given up in the merging process. Three different semantics of priorities are defined, two of which are equivalent. However, [125] does not consider negation (neither weak nor strong) at all, and the approach has thus limited expressive capability.

In [95], rules with negation in the head are considered as exceptions to more general rules, and they are given higher priority. Technically, this is achieved by a redefinition of answer sets. Answer sets as defined in Section 5.2 are also answer sets according to the definition of [95]. The main achievement is that programs which merely admit an inconsistent answer set (as possible in the definition of [77]), become consistent in the new semantics. Thus, the approach in [95] has more of a contradiction removal method than a priority approach.

For further discussion of all these approaches, we refer the reader to [35].

7 Revision and Update

An agent's knowledge base is naturally subject to change, which may happen through different events, including the following:

1. new information is arriving at the agent (this could be, e.g, the result of a sensing action, or a message from some other agent);
2. temporal evolution, e.g., a "tick of the clock" event activating some predefined evolution [13]; or
3. the tentative merging of the current knowledge base with another one.

Two basic approaches to this change process can be distinguished: *revision* and *update*. The idea of program revision is that of a program forced to include additional information about a world supposed to be static, by revising the assumptions it contains. On the other hand, the issue of program updating is to deal with a dynamic world [92]. A detailed discussion on the different approaches to changes of the knowledge base can also be found in [75, 116]; a clear distinction is not always possible, and "hybrid" formalisms are possible as well [60].

Moreover, a plethora of other features distinguish the different methodologies such that a clear, orthogonal classification is not straightforward. In particular, the following features can be found:

- Revisions and updates can involve two theories or two sets of rules and facts (Sections 7.1-7.5, 7.9, and 7.7), often two logic programs, or the current program or theory plus a single rule or fact to be inserted in it (Section 7.8). Some of them eventually consider the case in which a single rule or fact has to be retracted too (Sections 7.1-7.3).
- Update formalisms usually implicitly assign higher priority to the "new" information, thought to be the result of changes in the external world, which the agent should be forced to accept (Sections 7.1-7.8). Other formalisms assume no priority between old and new information; in this case, the term *merging* is often used (e.g., Section 7.9).

- As in the case of priority handling, some methods define an ad hoc semantics (Section 7.8), while others rely on a rewriting of the original programs under some standard semantics (Sections 7.2, 7.4, 7.5, and 7.7). If the update policy is explicitly expressed, changes to the process strategy (e.g., arbitrary priority assignment to the involved programs instead of fixed ones) could be easier to perform. Sometimes, the update policy may itself be subject to change (Section 7.8).

- Most of the approaches pose no explicit requirements on the consistency of the old or of the new knowledge base (Sections 7.1–7.3, 7.5). As a consequence of the update process, conflicts may sometime arise, and the formalism may implicitly contain a strategy for inconsistency removal leading to a consistent belief set (Sections 7.3, 7.4, 7.8). For other formalisms, addressing consistency requires changes in the update approach and its semantics (Section 7.5).

- In some approaches, old knowledge, which is overridden or contradicted by the new one, is physically deleted from the knowledge base (Section 7.1–7.3, 7.9), while in other approaches it is only temporarily de-activated (Section 7.8) and can be activated again by successive updates (Sections 7.5 and 7.7).

- It is also important to consider whether the formalism imposes some form of minimality of updates, in order to preserve "as much as possible" of the current knowledge when new one is to be incorporated. Some of the methods include already in their basic definitions some condition for minimality of change (Sections 7.1–7.4, 7.9), while for other formalisms additional limitations are required or no minimality notion is given at all.

- Some of the formalisms apply only to single updates (Sections 7.3, 7.8, 7.9), while others can be applied uniformly to single updates as well as to sequences of updates (Sections 7.5, 7.7). For some of the formalisms based on the rewriting of the involved programs iteration is possible, i.e., the program resulting after an update can be directly subject to update again. (Sections 7.1, 7.2, 7.4, 7.5, 7.8, 7.7).

In what follows, we discuss several update and revision formalisms in more detail.

7.1 Revision Programs by Marek and Truszczyński

In [112], a language for revision specification of knowledge bases is presented, which is based on logic programming under the stable model semantics. A knowledge base is in this context a set of atomic facts. Revision rules describe which elements are to be present (so-called *in*-rules) or absent (*out*-rules) from the knowledge base, possibly under some conditions. Note that such a definition is self-referential, because the latter conditions must be evaluated over the resulting knowledge base. A fixed-point operator, which satisfies some minimality conditions, is introduced to compute the result of a revision program. As for stable models, there may be several or no knowledge base satisfying a given revision program. While the approach was formulated for the propositional case, a possible extension to the predicate case is also briefly discussed.

7.2 Update Rules as Logic Programs by Pereira et al.

In [11], transition rules for updating a logic program are specified by means of another logic program. Update programs are defined as a collection of update rules, specifying all the atoms which must be added or deleted depending on some pre-conditions.

The notion of justified update of a total interpretation for an initial program is defined, given an update program and a *stability condition*, on the basis of revision definitions. The stability condition guarantees that the initial interpretation is preserved as much as possible in the final one, applying inertia rules to those literals not directly affected by the update program.

In case the initial base is a normal logic program (NLP), a transformation is given which produces an extended logic program (ELP) whose models enact the required changes in the models of the initial program, as specified by update rules. The transformation guarantees that, by inertia, rules in the initial program remain unchanged unless they are affected by some update rules. This is realized by renaming all the atoms in the rules which are possibly affected by some update rule, and introducing inertia rules stating that any possibly affected atom contributes to the definition of its new version, unless it is overridden by the contrary conclusion of a rule from the update.

A generalization of this transformation to the case where both the initial and the update program are ELPs is also given. This generalization is necessary because the result of the update is an ELP but not an NLP in general; thus, a sequence of updates could not be processed by simple iteration of updates. The approach is similar to the approach for NLPs and works as desired under the well-founded semantics. With this extension, successive update transformations may take place.

In later papers [100, 9, 10], the focus is shifted from models to programs. The idea is that a logic program encodes more than a set of models: it encodes also implicit relationships between the elements of those models. The principle of inertia should therefore be applied to the rules in the program rather than to the atoms in the models. In order to update a program, it must be checked whether the truth value of a body which determines the truth value of a head has not changed, before concluding the truth value of the head by inertia. In fact, the truth value of the body may change due to an update rule. The truth of any element in the updated models should therefore be supported by some rule (one with a true body) either of the update program or of the given one, in the light of the new knowledge.

¿From this point of view, model updates are a special case of program updates, since models can be encoded as factual programs. The updating of NLPs and ELPs is thus defined with the same approach, only shifting the application of inertia from atoms to rules in the original program. The resulting updated program depends only on the initial program and update program, but not on any specific initial interpretation. The rules of the initial program carry over to the updated one, due to inertia, just in the case they are not overruled by the update program. This is achieved by defining the subprogram of the initial program which contains the rules that should persist in the updated program due to inertia. This subprogram is then used together with the update program to characterize the models of the resulting updated program.

A joint program transformation is thus given, taking the initial and the update program (which introduces also inertia rules), and producing an updated program whose models are the required updates.

7.3 Abductive Updates by Inoue and Sakama

In [88], Inoue and Sakama present an approach to theory update which focuses on nonmonotonic theories. They introduce an extended form of abduction and a framework for modeling and characterizing nonmonotonic theory change through abduction. This framework is shown to capture and extend both view updates in relational databases and contradiction removal for nonmonotonic theories.

Intuitively, this is achieved by extending an ordinary abductive framework to autoepistemic theories [115], introducing the notions of negative explanations and *anti-explanations* (which make an observation invalid by adding hypotheses), and then defining autoepistemic updates by means of this framework.

The aim of the extension is to provide abduction with the following features:

- both the background theory and the candidate hypotheses may be autoepistemic theories;
- hypotheses can not only be added to the theory but also discarded from it to explain observations;
- observations can also be unexplained.

Inoue and Sakama introduce the notion of a minimal pair of positive and negative (anti-)explanations, and define update as a function on autoepistemic theories, which consists in the insertion and/or deletion of formulas in autoepistemic logic, based on (anti-)explanations accounting for the new observations. A combination of such an abductive and inductive framework is further described in [134].

The framework of extended abduction is then used in [90] to model updates of nonmonotonic theories which are represented by ELPs. Using appropriate update programs, i.e., appropriate ELPs specifying changes on abductive hypotheses, the following problems are solved through abduction:

View updates: The problem of view update consists in incorporating new literals in the variable part of a knowledge base in which variable and invariable knowledge are distinct (as in a database made up of an intensional and an extensional part, updates to the intensional part have to be translated into updates to the extensional part, the variable one).

Theory updates: For theory updates, the whole knowledge base is subject to change. New information in form of an update program has to be added to the knowledge base and, if conflicts arise, higher priority is given to the new knowledge. The updated knowledge base is defined as the union $Q \cup U$ of the new information U and a maximal subset $Q \subseteq P$ of the original program that is consistent with the new information (which is always assumed to be consistent). The abductive framework is in this context used for specifying priorities between current and new knowledge, by choosing as abducibles the difference between the initial and the new logic program.

Inconsistency removal: The problem of inconsistency removal is seen as special case of theory update, where the initial program is inconsistent and the new program is empty.

These three different problems are considered in a uniform abductive framework, namely that of [88]. Based on the necessary update rules, an appropriate update program is defined for each case, which is an ELP specifying the changes on the abductive hypotheses.

7.4 Updates by Means of PLPs by Foo and Zhang

In [69], the update of a knowledge base of ground literals by means of a prioritized logic program (PLP, as defined in Section 6.4) over the same language is described. An update is itself a PLP over an extended language, consisting of the following elements:

- a program P containing initial knowledge rules, inertia rules and update rules;
- a naming function N for rules in the resulting program; and
- a partial order $<$ containing a pair $r < r'$ for each inertia rule r and each update rule r' in the new program, stating higher priority of inertia rules.

The possible resulting knowledge base is defined on the basis of the answer sets of the resulting program as follows: if it has no answer set, then the updated knowledge base coincides with the initial one; if its answer set is the set of all literals in the extended language (i.e., if we have inconsistency), then the updated knowledge base is given by the set of ground literals of the original language; if it has a consistent answer set, then the updated knowledge base is given by the set of ground literals of the original language such that the corresponding literal in the extended language belongs to the answer set of the resulting program. It is shown that the introduced definitions satisfy the minimal change property with respect to set inclusion.

In [70], the problem of updates when both old and new knowledge are encoded in ELPs is then addressed. The idea in updating the initial program with respect to the new one is to first eliminate contradictory rules from the initial program with respect to the new one, and then to solve conflicts between the remaining rules by means of a suitable PLP.

In the first step, each answer set of the initial program is updated with the information in the new one, producing a set of ground literals which has minimal difference to the answer sets of the initial program and which satisfies each rule in the new one. If the resulting set of literals is consistent, a maximal subset of the initial program is extracted such that the given set of literals is coherent with the union of this maximal subset and the new program. Thus the given set of literals is a subset of an answer set of the defined union. The resulting subset, called *transformed program*, is guaranteed to maximally retain rules of the initial program which are not contradictory to rules of the new program. If the given set of literals is not consistent, then the transformed program is any maximal subset of the initial program such that its union with the new program is consistent.

In a second step, possible conflicts between the rules in the transformed program and in the new program need to be solved. To this purpose, a special PLP is introduced as follows:

- the program itself is the union of the transformed program and of the update program;
- a naming function for rules is given;
- the priority relation contains all the possible pairs consisting of one rule of the transformed program and one rule of the new program, stating higher priority of new rules.

The semantics of the update is thus given by the semantics of the corresponding PLP.

7.5 Dynamic Logic Programming by Alferes et al.

Dynamic logic programming (DynLP) is introduced in [9] as a new paradigm, which extends the idea of an update of a logic program by means of a logic program (denoted $P \oplus U$) to compositional sequences $\bigoplus P = P_1 \oplus P_2 \oplus ... \oplus P_n$.

The result is obtained by defining a new extended language and building up the program update $\bigoplus P$ over this new language by means of a (linear-time) transformation of the set of rules contained in the original sequence to a generalized logic program (GLP), which encodes the intended sequence of updates. An extended interpretation is defined from interpretations over the original language to interpretations over the new one. The stable models of the program update are defined to be the projection of $\bigoplus P$ to the original language, and a characterization of stable models in terms of rule rejection set is provided.

In [10], some further features of DynLP are discussed. The first one is the efficient representation of background knowledge, that is to say some kind of knowledge that is true in every program module or state. Background rules are easy to represent and efficient to handle, as adding a rule to every state up to the current one is equivalent to adding that rule only in the current state.

Another issue for which several possibilities are mentioned is dealing with contradiction and how to establish consistency of updated programs. The contradiction may be explicit and may depend on several reasons: the current state is already inconsistent or the update program is contradictory, or both. In the first case, an approach to deal with this situation is to prevent the contradiction to be inherited, by limiting the inheritance by inertia. In the second case the solution is not so intuitive, as the rules of the update program must be, by definition, true in the updated program. A possible approach here is to require the update program to be consistent. An alternative approach is to accept contradiction in the current update and again prevent it from further proliferating by changing the inertia rules as in the first case. In the third case, the same approach is possible as well, or it is possible to establish priorities between different rules in order to prevent contradiction.

Yet another kind of contradiction is the implicit one: when the updated program is normal, explicit contradiction cannot arise, but it may well be that the updated program has no stable model. A possibility consists in shifting to three-valued stable semantics or well-founded semantics.

The approach suggested suffers from problems with changing the inertia rules. While it is easy to encode several conditions and applications to prevent or remove contradiction, to enact preferences, or to ensure compliance with integrity constraints, such changes in the inertia rules require that the semantic characterization of the program updates is adjusted accordingly. This seems far from trivial.

DynLP is combined in [55] with agents based on the paradigm of Kowalski and Sadri [96], for building rational reactive agents which can dynamically change their knowledge bases and goals (directed by observations), and learn new rules. According to the approach of Kowalski and Sadri, each agent is an abductive logic program executing an *observe-think-act cycle*, where abducibles are given by actions to be executed or explanations of observations. Inputs to each cycle are the required updates; for each agent and for each cycle the knowledge base of that agent is identified with the dynamic program update at the corresponding state.

7.6 Updates and Preferences by Alferes and Pereira

In [12], an integrated framework combining updates and preferences is introduced, based on DynLP and on the ideas for a semantics of preferences in [35].

A new language is defined, modeling sequences of programs $P_1, \ldots, P_n$, resulting from the consecutive updates of an initial program, together with a priority relation among the rules of all successive programs. The priority relation is itself subject to update.

This integrated approach is based on the idea that both preferences and priorities eliminate rules: preferences eliminate less preferred rules, to select among the available stable models, and updates eliminate rules overruled by other ones, in order to generate new models. Preferences require a strict partial order on rules, while updates require a linear temporal order, or other distinct linear structures, allowing nevertheless the production of a tree of linear updating sequences.

In this framework, preferences may be enacted on the results of updates, whereas updates may be used for the purpose of changing preferences. Preferences are under this view intended to select further rules after computing the results of updates. Intuitively, starting from the semantics of updates (erasing rejected rules), the semantics of preferences is defined (erasing unpreferred rules) as required in [35] and aiming at a combination of the two. An integrated semantics is then formulated for both of them.

The object language of the approach is that of DynLP under the stable model semantics; explicit negation can be expressed by corresponding new rules and atoms. Formally, these prioritized dynamic programs are defined using a strict partial order over rules. This preference information is used to prefer among stable models which are a fixed-point of an equation guaranteeing that the rules are being applied in observance of the partial order.

In order to facilitate the capturing of both preferences and updates in a single framework, the semantics has to be based on the removal of less preferred rules. The main issue is therefore to determine criteria for rules to be removed in order to obtain the same result as in [35]. This is achieved in the following way: while in [35] the head of a rule is not added to the construction of a preferred stable model if the rule is defeated by the previous constructed set (formed by the heads of the more preferred rules), the same effect can be obtained by removing all rules defeated by the head of a more preferred rule which in turn has not been removed itself. In other words: remove all rules whose body is in the model and whose head defeats a more preferred rule. If the body of a less preferred rule is not actually true in the model, then its defeat is only potential and the rule must not be eliminated. Formally, the notions of unsupported and unpreferred rules are introduced, in order to obtain the appropriate definition of the preferred stable models. The definition of preferred stable models is equivalent to the one of preferred answer sets given in [35].

After defining the appropriate semantics of preferences, preference and update semantics are combined. The fundamental question is where to define the priority relation. Two possible levels for the definition of priorities are identified: (i) priorities among rules of the same program, or (ii) priorities among rules in the union of all programs in the sequence. According to [12], the second possibility is more general, as it does not prevent limiting the priority relation to rules in the same program. The chosen language for such a representation must be able to cope at the same time both with evolution by means of updates and evolution of the priority relation at the same time, and is an extension of DynLP: instead of sequences of GLPs $P_1, \ldots, P_n$, sequences of pairs $\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle$ are given, where each pair $\langle P_1, Q_i\rangle$ consists of a program P_i representing rules and a program Q_i describing a priority relation among rules. In general, as an update of the priority relation may depend on some other predicate, rules in programs describing priorities are allowed to refer to predicates defined in the programs representing knowledge.

Besides the extension of DynLP to consider preferences as well, the work in [55] about updating agents is extended for preference and updating in multi-agent systems [54]. Preferences are thus included among the rules of an agent and allow a selection among several models of the corresponding (updated) knowledge base. Preferences can themselves be updated, possibly depending on information coming from other agents.

7.7 Inheritance Programs and Updates

Referring to the approach presented in Section 6.6, it is natural to view a sequence of updates as an inheritance program where later updates are considered more specific. That is, we might view the sequence of programs $P = P_1, \ldots, P_n$ as a knowledge base with inheritance $P_n < P_{n-1} < \cdots < P_2 < P_1$, i.e., for each $j = 1, \ldots, i$, P_{j+1} is more specific than P_j [60]. The corresponding inheritance program is equivalent to the update sequence $P_1, \ldots, P_n$ for certain classes of programs [61].

As shown in [10], dynamic logic programs naturally generalize the idea of updating interpretations, viewed as collections of facts A and $not\ A$, respectively, through Marek and Truszczyński's revision programs [112], which have been further considered by Przymusinski and Turner [127]. Thus, inheritance programs also generalize interpretation updates.

Buccafurri et al. describe in [38] a transformation from disjunctive inheritance programs to inheritance-free disjunctive programs, similar in spirit to update programs (cf. Section 7.5).

7.8 Revision of Preference Default Theories by Brewka

In [34], Brewka introduces a nonmonotonic framework for belief revision which allows reasoning about the reliability of information, based on meta-knowledge expressed in the language of the information itself. In this language, revision strategies can be declaratively specified as well. The idea is to revise nonmonotonic theories by adding new information to the current theory, and to use an appropriate nonmonotonic inference relation to compute the accepted conclusions of the new theory. The approach is described in Section 6.7.

The main advantage of this approach is that it allows the explicit representation of preference information, which is commonly used by agents in the process of revising their beliefs. Once preference relations are represented in the language itself, it is possible to represent revision strategies declaratively as well, and to revise them too.

In this context, epistemic states are identified with preference default theories, and belief sets with their accepted conclusions, since preference default theories under the accepted conclusion semantics always yield consistent belief sets. The belief sets cannot be revised directly, but only through the revision of the corresponding epistemic state. Given an epistemic state, revising it with new information simply means generating a name for the new information, and adding it to the theory.

The notion of preferred extensions of the revised theory serves to compute the revised belief set, taking into account the newly added information, possibly containing preference information as well.

The suggested notion of revision is evaluated with respect to the AGM postulates [7] (suitably reformulated in terms of belief sets), and it is explained why it is meaningful that some postulates are not satisfied by the approach.

Contraction, i.e., making a formula underivable without assuming its negation, is implicitly handled as well, as far as the agent receives information of the form *"believe* $\neg p$*"*. However, if the form of the incoming information is *"do not believe* p*"*, the introduction of an extra mechanism seems necessary to make the formalism able to deal with it. To this purpose, the idea of distinguishing constraints from premises can be useful: constraints are viewed as formulas used in the construction of maximal consistent subsets of premises, but are not used in derivations to compute the deductive closure of a set of formulas. The compatibility of the extensions with the preference relation has then to be checked against both constraints and premises, whereas extensions are generated only from the premises.

In this framework, knowledge bases grow monotonically. However, for agents with limited resources, the expansion of the knowledge base cannot go on forever. The problem consists in determining how and when some pieces of information should be forgotten, activating a sort of garbage-collection strategy. This requires a sort of utility notion over information.

7.9 Arbitration

A problem which is closely related to revision and update is arbitration between two or more knowledge bases, which is typically performed when they are to be merged into a single knowledge base. Under arbitration, one understands combining the knowledge of different knowledge bases having the same priority.

Approaches from logic programming, such as inheritance programs [39, 38] and extensions of dynamic logic programming [10, 8], may be used to technically address this problem, even though they have not been specifically designed for arbitration between knowledge bases. Thus, they may not lead to satisfactory behavior, and, in particular, simple inconsistencies between knowledge bases of which each one is consistent (e.g., $\{a \leftarrow\}$ and $\{\neg a \leftarrow\}$) may lead to inconsistency of the combined knowledge base. On the other hand, ordered logic programming [98, 72, 37, 40] can avoid this problem. The notion of a stable model [37, 40] is in that context similar to the notion of an answer set, once conflicts between rules have been eliminated.

Thus, it can be seen as a kind of answer set with implicit contradiction removal between knowledge bases. While ordered logic does not provide weak negation, it can be simulated in the language [40].

Observe that specific approaches for combining classical propositional and first-order knowledge bases have been defined in [25, 130, 26, 106], while [27] has addressed the problem of combining default logic knowledge bases which must satisfy given integrity constraints. ELPs inherit this method by the correspondence between ELPs and default logic theories. Briefly, in terms of ELPs, the approach tries to retain as much as possible from the union of the sets of rules in the programs $P_1, \ldots, P_n$ which are considered, such that consistency is retained and all integrity constraints are satisfied. At the heart of the approach is a contradiction removal method for conflicts that arise between consistent default theories of equal priority. The authors also extend their method to prioritized combination of default theories.

Axioms for arbitration operators in the context of classical propositional knowledge bases have been addressed in [130] and [106]. In [130], arbitration is defined as a model-fitting operator, and it is shown that the axiomatic description of arbitration operators is incompatible with revision operators as in AGM theory [7] and update operators according to the theory in [92]. On the other hand, in [106], axioms for arbitration operators are considered such that, as stated, belief revision can be easily formulated through arbitration but, on the contrary, reformulation of arbitration in terms of belief revision is in general much more complex.

We finally remark that, like for update and revision operators, no axiomatic principles (except rather obvious ones) in the style of AGM revision theory have, to the best of our knowledge, been discussed so far for arbitration operators between logic programs. The discussion which properties such operators should enjoy is at an early stage, and further research in this direction is needed.

8 Quantitative Information

Since van Emden's pioneering paper [139], the problem of expressing quantitative information in logic programming has been addressed in several formalisms, depending on the meaning and the nature of the quantitative information to be represented, and on the class of reasoning tasks to be performed.

Some formalisms allow the quantitative information to be associated with rules (e.g., [113]), others with rule heads or rule bodies (e.g., [121, 109, 110]), or with single atoms of the language (e.g., [117]), or with formulas (e.g., [118]). The information is sometimes expressed by means of a single value (e.g., [113]), sometimes in the form of ranges or intervals of values (e.g., [118, 119, 121, 109, 110]), maybe allowing variables to appear in the definition of such intervals.

The point is that quantitative information has been considered for addressing different problems, which impose different requirements, and thus lead unsurprisingly to different solutions and suggestions. Quantitative information can

- express preferences and priorities by means of an absolute value instead of an order over rules;
- be related to the values and ranges assumed by numeric variables, or to probability distribution over domains;
- express costs and weights of performing some operations, thus implicitly encoding preference information;
- express uncertainty and probabilities of events or of beliefs;

- originate from statistical information, and be subject to further updates, and so on.

We discuss here only a few approaches, which are quite different in their main aims and theoretical foundations.

8.1 Disjunctive Programs with Weak Constraints by Buccafurri et al.

An enhancement of disjunctive logic programs by means of constraints is presented in [41]. Two basic types of constraints are considered: *strong* and *weak* constraints. Strong constraints must be satisfied like ordinary integrity constraints in logic programming, whereas weak constraints are, if possible, to be satisfied, and violations are taken into account when selecting the preferred models of a program. Intuitively, the suggested semantics minimizes the number of instances of weak constraints violated by a model. Moreover weak constraints may be assigned different priorities, resulting in a ranking of the models of the program.

This extension of disjunctive programming is especially useful to encode planning problems, combinatorial optimization problems, and abductive reasoning mechanisms, since candidate solutions can be checked for mandatory properties through strong constraints and then the best solutions can be chosen minimizing weak constraint violations.

8.2 Weight Constraint Rules by Niemelä et al.

In [121], logic programming is extended in order to allow quantitative information in the body of rules. The extended rules are particularly useful for representing combinatorial optimization problems, to encode constraints about cardinality, costs and resources, which otherwise would be quite cumbersome or impossible at all to encode in usual logic programming. The suggested semantics is shown to generalize the stable model semantics for NLPs [76].

In order to obtain the desired expressiveness, cardinality constraints are first introduced, which are satisfied by any model such that the cardinality of the subset of literals involved in the constraint and which are satisfied by the model lies between the specified lower and upper bound for that constraint. This kind of constraint is then generalized to the first-order case, and real-valued weights over variables are introduced, giving the constraints a meaning of linear inequality. Finally, weight rules involve in their body a set of such weight constraints over a set of variables.

8.3 Weighted Logic Programming by Marek and Truszczyński

In [113], weighted logic programs are introduced. Each rule in the program is assigned a non-negative real number called *weight*, which is interpreted as the cost of applying that rule. Two different approaches for the computation of costs are discussed, called respectively *reusability* and *no-reusability*. In the reusability approach, the cost of applying rules for deriving an atom is paid only once, and then the derived atom can re-used as often as needed; in the no-reusability approach, the derivation cost has to be paid each time an atom is needed to be used in the body of a rule.

In this context, an interesting problem is the computation of the minimal cost for deriving an atom or a set of atoms. In the no-reusability approach, for the propositional case the minimum cost of a derivation can be computed by polynomial-time shortest-path algorithms. In the reusability approach, computing the minimal derivation cost is NP-hard.

8.4 Probabilistic Programs by Subrahmanian et al.

In a sequence of papers [118, 119, 120, 51], a theory of probabilistic logic programming is introduced and developed, based on the works about annotated logic [135, 93], and on the possible-world semantics. Alternatively, probabilities can refer to the elements in the domain, and represent statistical knowledge. We do not discuss this approach here. For more information on this topic, see e.g. [117, 124, 22, 123].

In the possible-worlds approach, a probability range can be assigned to each formula in the body and in the head of a rule, and can contain in its definition variables ranging over $[0, 1]$. A suitable model theory and semantics is introduced for negation-free probabilistic logic programs [119], which is then extended to handle default reasoning under the stable model semantics [120]. This approach to probabilistic logic programming is used in [58] to extend the framework for agent programming presented in [31], in order to express probabilistic information deriving from the lack of knowledge of the agent about its effective state. For example, an agent can identify an object in its environment with some degree of uncertainty, which must be considered in its decision process. This idea is extended further in [59] in order to handle probabilistic beliefs: probabilities can be assigned to agents' beliefs about events, rather than to events themselves.

An alternative approach to probabilistic logic programming with possible-world semantics is discussed in [109, 110], where probabilistic rules are interpreted as conditional probabilities. Program clauses are extended by a subinterval of $[0, 1]$, which describes the range for the conditional probability of the head of the rule, given its body.

9 Temporal Reasoning

An important issue for information agents is temporal reasoning, which is less investigated in the context of declarative logic programming. In particular, reasoning about the temporal evolution of the agent's knowledge base and its behavior, depending on an underlying update policy or other principles, are of major interest. This poses particular requirements both on the knowledge representation language, which must allow to express temporal features, and on the reasoning mechanisms.

The topic may be considered from different perspectives. Temporal aspects are relevant when considering agent actions and their effects, also in conjunction with beliefs, desires and intentions hold by the agent. On the other hand, the same temporal aspects need to be considered for reasoning about knowledge updates, as described in Section 7. This approach has just been started with the definition of LUPS [13], a language for declarative specifications of the evolution of a knowledge base. In the following subsections, we briefly discuss some works in the field of action languages, temporal logics for BDI agents, the language LUPS, as well as a pointer concerning how temporal logics and model checking techniques could be useful in conjunction with LUPS for reasoning over updates.

9.1 Reasoning about Actions

The agent needs to reason about direct and indirect effects of its actions, possibly involving several temporal units belonging to future time states. Effects may be nondeterministic, requiring the agent to take into account concurrency [23] and ramifications [82]. In this field, much work has been spent on describing and discussing the features of alternative action languages [78, 107], together with the corresponding reasoning mechanisms, combining them with temporal action logics or with the situation calculus [83]. A review of the relevant tasks for reasoning agents in such dynamic domains can be found in [24].

Actions may be related to the update of a knowledge base in different ways. An update process can be modeled as an action, which takes as input the update information and, as an effect, incorporates this information into the knowledge base. The difference, however, is that this action changes the intensional part of the epistemic state, while usual actions only change the extensional part. Vice versa, a proposal for expressing actions as updates was given in [14].

9.2 Temporal Logics for BDI agents

Temporal aspects are addressed in BDI agents (*Belief*, *Desire*, *Intention*; see, e.g., [81]) by means of an extension of the branching-time temporal logic CTL* [65]. This approach is based on a possible-worlds model, represented by a time tree with a single past and a branching-time future. Branches in the time tree represent the different choices available to the agent at each given point of time, as to which action to be performed. A set of belief-accessible worlds is associated with each status, namely the worlds that the agent believes to be possible. Each belief-accessible world is in turn itself a time tree. Multiple worlds result from the agent's lack of knowledge about the effective state of the world, and within each of these worlds the different futures represent the choices available to the agent in selecting an action to perform. The same approach is used for goal-accessible and intention-accessible worlds as well. In order to handle these possible-worlds, CTL* is extended by introducing modal operators for beliefs, goals and intentions, and by adding first-order features.

9.3 LUPS, a Language for Specifying Updates

Alferes et al. introduce in [13] the language LUPS for specifying update sequences of logic programs. The need for such a language originates from the consideration that dynamic logic programming describes how updates are to be performed, but offers no possibility for specifying the content of each update program in a sequence. Given the current epistemic state, how to (declaratively) specify the appropriate updates to be performed? How to describe rules and laws for the update behavior of an agent? LUPS statements specify how a logic program should be updated, describing which rules under what conditions should be incorporated into or retracted from the knowledge base. In the space of the epistemic states for an agent, a LUPS program constitutes a transition function from one state to another state.

The semantics of LUPS is formally defined by means of the semantics of the corresponding DynLP. A translation of a sets of LUPS statements into a single GLP, written in a meta-language, is provided, so that the stable models of the resulting program correspond to the intended semantics. Knowledge can be queried at any epistemic state.

10 Evaluation

In the light of the general features of information agents, as identified in Section 2, and of the problems and challenges as discussed in Section 3, we may evaluate the declarative methods reviewed in Sections 6 and 7. Criteria for the evaluation are:

- what kind of problems and challenges for information agents a method can address;
- how difficult it is to modify a method for slightly different purposes or policies, to generalize or extend it, or to couple it with other standard methodologies;

- what limitations and bounds a method poses on the objects it processes;
- how near or similar to other formalisms a method is;
- which of the possible features identified for the tasks in Section 3 can be considered as desiderata over corresponding formalisms, and which are satisfied by the method in question.

As far as temporal reasoning over evolution of updates is concerned, a possible approach to this task is discussed in Section 11.

10.1 Preference handling

Some requirements or desiderata can be identified with respect to formalisms for preference handling, which allow some evaluation of the methods we reviewed.

First of all, it must be possible to express priorities over single rules and facts as well as over sets of them. This is required in order to select the information sources to answer a given query, to account for the reliability of the sources, or to update the knowledge base of the agent with some information coming from an external source. The update can be coupled with priority information, e.g., recording the accuracy or reliability of the new knowledge. Moreover, priority information makes it possible to merge answers from different sources into a single query answer which should be forwarded to the user, and gives a powerful instrument to remove possible inconsistencies and select the intended answers.

¿From the point of view of priority level (over rules or over sets of rules), the approach of Section 6.3 is the only one which considers priorities over literals. This seems not necessarily the best solution for information agents. It allows, however, to indirectly express also preferences over rules by means of additional atoms and corresponding rules.

If possible, preferences should be expressed in the object language of the knowledge base, by means of a naming function for rules and of a special symbol in the language, so that priorities can be dynamically handled.

Dynamic priorities could be of interest for addressing several of the problems for information agents. E.g., the reliability of an information source may not be fixed in advance, but depend on the field of interest, or a query answer may depend on the user profile. Consider, for example, the adaptive Web site of a clothing company, which determines what items have to be shown depending not only on the explicit input from the user (who is looking for brown shoes, say), but also on the average value of the orders of the customer, and then shows the selected items ranked after their price. By means of dynamic priorities, the items more likely to be interesting for each customer can be better selected, and different ranking criteria may be adopted for different customers: e.g., customers usually buying expensive articles may be presented with expensive items first (in order to maximize the profit, in case the customer decides to buy something), while customers who in the past preferred cheaper items may be presented a list in inverse order, to encourage them to buy.

Concerning the possibility of addressing preferences over preferences, methods with explicit priorities in the knowledge base itself are to be preferred, like the ones in Sections 6.1, 6.5, and 6.7.

A mechanism is then needed to compute the intended set (or sets) of conclusions, starting from the logic program expressing the epistemic state of the agent. Priorities in the program can be thought of as constraints on the construction of this set, and determine which of the conclusion sets (like stable models or answer sets) is to be preferred. This is obviously useful to merge answers coming from different sources, and to resolve possible inconsistencies. Mechanisms based on standard semantics rather than on ad hoc semantics offer some advantages:

1. they are easier to be compared to other approaches;

2. their behavior can be better understood;

3. they are easier to be integrated or extended to other frameworks.

Particularly useful are the methods based on some rewriting of the (prioritized) logic program, since changes to the rewriting algorithm can implement other kinds of preference handling in the same framework, see Sections 6.1, 6.5, and 6.6.

The policy for priority handling (determining e.g. which rules are to be considered conflicting and which ones are to be given precedence) should be possibly explicitly encoded in the object language of the knowledge base, in form of rules about preferences. This is the case in the approaches based on a rewriting of the program, as well as in the method of Section 6.7. This makes it possible to implement in the same program more than one of such policies, to be used in different situations. E.g., the way in which priorities are to be handled to determine a user's profile may be different from the way in which they are to be handled to choose the most promising information source.

It would also be useful to have the possibility of expressing several levels of priorities. Consider for example the situation in which new information (in form of a single rule or a set of rules) comes from another agent, and is to be incorporated in the knowledge base: a priority value expresses the general reliability value for that agent, and single values express e.g. the reliability of each rule with respect to the domain of interest.

This feature is also interesting for the application of learning procedures on the basis of the history experienced by the agent. Different levels of priorities could undergo different learning and updating policies, making lower levels easier to modify, and requiring more striking evidence before changing priorities for the higher levels.

None of the methods we discussed offers such a possibility, but some of them could be easier extended than others, in particular the approach in Section 6.5, and perhaps the one in Section 6.6.

As regards the flexibility of the approaches, some of them are interesting since they allow to capture other formalisms as well, e.g. the approaches discussed in Sections 6.3 and 6.5. An interesting aspect is the possibility of unifying preferences and updates in a single framework. A tentative approach is reviewed in Section 7.6 and discussed in Section 10.3. Another approach consists in adding the update knowledge to the current one in the form of a set of rules with higher priority. This is possible e.g. in the approach reviewed in Section 6.1, as well as in the one of Section 6.5. A first comparison of the latter one with the update policy of Section 7.5 and [60] reveals that the semantics is different, but further investigation is needed to elucidate the relation.

¿From this point of view, an interesting issue is addressed in [90], concerning a form of reasoning about preferences in which "sufficient" preferences to derive intended conclusions are abductively selected. In the other formalisms we considered, no similar form of reasoning is suggested. This feature could be particularly interesting from the point of view of learning.

Further improvements should also provide the possibility of associating quantitative information with rules, in contrast to relative preference information. General principles for preference handling (as in [35]) and particular properties of the approaches as to the behavior of the corresponding agent should be better analyzed. Interesting is also the problem of self-reference of the preference relation pointed out in [34], which has not been discussed in other approaches.

10.2 Logic Programs with Quantitative Information

If quantitative information is to be incorporated into logic programs, the first question to be answered is what kind of information it is, and where this information should be added. Several possibilities have been suggested, each of them addressing a different purpose and usage of quantitative information:

Rule level: Quantitative information may be associated with rules in the form of a single number, expressing preference or priority by means of absolute values instead of relative order among rules. For example, this is the case in weighted logic programming [113] and also, to some extent, in disjunctive logic programming with weak constraints [41], where a ranking of the constraints is possible. This feature can be useful if the agent has to solve combinatorial problems (e.g., resource allocation).

Rule bodies: Another approach useful for combinatorial optimization consists in associating quantitative information with the bodies of the rules, in form of a range identified by lower and upper bounds [121].

Atom level: The quantitative information can be associated with atoms in form of a probability range, or with formulas. In case of non-atomic formulas, the question arises as to how the probability ranges associated with the underlying atomic formulas have to be combined, or, vice versa, decomposed. In different application domains, there may be different dependency relationship between atomic formulas. An interesting solution is discussed in [51], where the user is allowed defining and using different operators in the same probabilistic logic program, to combine ranges, corresponding to the different dependency relationships which may exist between formulas (e.g. mutual exclusion, independence, and so on). Probability information can be successfully used e.g. in problems of user profiling, in the tentative integration of incomplete information, and in learning procedures for agents. An implementation of this proposal is already possible in the IMPACT environment, as described in [58], whereas the behavior in practical domains of such probabilistic agents has to be further investigated.

Conditional probability: If the quantitative information expresses a conditional probability between the head and the body of a rule, the probability range is to be associated with the head of rules, as suggested in [109]. Applications of this approach in real-world domains should be studied, since it could be interesting to solve the problem of guessing missing pieces of information (e.g., in a user profile or in a query answer), or how to choose among conflicting conclusions.

Most of these approaches require extensions of some standard semantics (often the stable model semantics), and the integration of these features in other logic programming frameworks seems to be difficult. For example, how should a sequence of probabilistic logic programs or of weighted logic programs be updated? How can strategies of inconsistency removal be defined, when the inconsistency between rules resides in the quantitative information associated with them or with part of them? Such questions need to be addressed, before information agents can take advantage of these formalisms.

10.3 Revision and Update

The aspect of revision and update has to be addressed in order to make information agents adapting their behavior over time, as external conditions change. For example, when an information source is no longer available, when the contents of a database or a user profile have been updated, or when a message from another agent in the same system has been received, and so on.

A correct knowledge update can be relevant in order to decide which information source has to be queried, or to modify an existing query plan to adapt it to run-time information. It has a great impact also when implementing learning procedures for the agent. In this case, changes and updates of the knowledge base are not only determined by external evolution over time, but also by internal processing.

The problem for information agents is the following: new information in form of a single rule or fact, or in form of a logic program, should be incorporated in the current knowledge base. The new information may represent a temporal evolution of the epistemic state, possibly originating from other agents who hold more reliable beliefs, or coming from the environment and communicate changes of content or availability of some database, or the information may be the result of some learning procedure over past experience, and so on. When incorporating this new knowledge into the current one, the consistency of knowledge and the correctness of the agent's behavior have to be further ensured.

The approaches which provide the possibility of updates by means of logic programs (cf. Sections 7.2, 7.5, and 7.7) are in our opinion to be preferred, since they can be applied in case of a single rule or fact as well. Furthermore, assuming that the knowledge base is represented by a logic program, no special language is required to specify updates, and the two logic programs can be used directly in the update process.

In the update process, it should be possible to merge the current with the new knowledge, giving a higher priority to the new one, thus solving possible conflicts arising during the merge process. This is possible in all the approaches we analyzed except arbitration (Section 7.9), which considers knowledge bases having the same priority. It would also be useful to have the possibility of explicitly specifying which of the knowledge bases to be merged has higher priority, since this would permit to model not only temporal evolution, but also reciprocity of update among agents. For example, information agents working on the same databases can exchange information as to the current availability of the sources, their average response-time, possible changes in their contents or available services, and so on. This is not addressed by the methods we discussed, as they focus on sequential updates instead of general priorities, and the common assumption is that the new information must be preserved by the update, maybe rejecting some old beliefs. For further considerations about possible handling of priorities, and about multi-level priorities, cf. Section 10.1.

Like for priority handling, those revision approaches are to be preferred whose update strategy is realized under a standard semantics, e.g. by means of a rewriting of the two logic programs (e.g. Section 7.5 and 7.7, together with the proposal in [60]). The rewriting technique makes the update policy explicit in the resulting knowledge base, and can be usually modified or extended in order to encode other strategies or update mechanisms. This could be interesting in order to differentiate the update process on the basis of the source of the update information, like, e.g., by another information agent, or by one of the data sources communicating changes in its content, or by internal learning procedures. The approach would also permit, as outlined in Section 10.1, a uniform handling of preferences and updates as a particular form of preference mechanism.

A tentative approach to this uniform framework is the one described in Section 7.6, which however suffers from some drawbacks, e.g., preference and update information are treated separately in pairs of programs. It is evident that in many application domains such a distinction is not possible. Consider an agent which obtains from an information source some data, together with their reliability, and the agent has to incorporate this new knowledge in its current epistemic state. In this case, the update information cannot be simply separated from its "priority", and the update process has to consider both aspects (in the current knowledge base this information may already be present, but with different reliability). Moreover, in the suggested framework [12], new information is always supposed to have higher priority, and only in a second step the preference information contained in the second program of each pair is considered to select further among the rules who "survived" the update. A more general approach would be useful.

Another method which could be used for the uniform handling of updates (as explained in Section 7.7) and preferences is disjunctive logic programming with inheritance [38], although it is originally conceived as a mechanism for representing inheritance hierarchies, thus not being appropriate for every sort of preference relation (cf. considerations in Section 10.1).

Other approaches addressing both updates and preferences have not been suggested yet. A possibility would be to extend an approach such as the one in Section 6.5, in order to handle updates as a special case of priority (as it actually is), but the current version of the rewriting mechanism produces for updates not always the intended results, compared with the approaches in Sections 7.5, 7.7 and [60]. Further investigation on this topic is required.

Another interesting issue lies in the possibility of updating the update (or revision) policy itself. This is only possible in the approach discussed in Section 7.8, in which the revision policy is declaratively encoded in the knowledge base. This feature would be very interesting for learning agents.

Other interesting features which are relevant to the process of updating are the following:

Inconsistency removal. Update methods are implicitly faced with the problem of inconsistency removal, in order to resolve conflicts, usually in favor of the new knowledge. Some methods address the topic explicitly, like the abductive updates of Section 7.3. Others can result in an inconsistent knowledge base (e.g. Section 7.5), and additional mechanisms or changes to the update process are required to ensure at least that inconsistency is not propagated to future updates. Ideally, an inconsistency removal strategy should be declaratively encoded in the knowledge base, as it is realized for preference and update policies. This allows of course different treatments of inconsistencies depending on their level and relevance. For example, a conflict in the "core" part of the knowledge base must always be resolved, while a conflict in a query answer for the user can be communicated to the user as a form of incomplete information, if the agent is unable to decide which of two data items is more reliable.

Retraction. As in theory revision, the problem of retracting a fact, a rule, or a set of rules should be addressed too. In the approach of Section 7.8, this is partially possible, if the retraction can be expressed in a "positive" form, while it is not possible for general retraction. Explicit retraction of atoms is possible only in the approaches of Sections 7.1, 7.2, and 7.3, whose languages allow the explicit definition of the atoms to be inserted or to be true in the resulting belief set and of the atoms to be deleted or false. For the other methods, this is not possible at all. It is evident that this aspect has to be addressed for e.g. learning procedures, as well as for some forms of communication and update exchange between agents.

Minimality of change. The agent should be forced to give up as little as possible of its past epistemic state when incorporating new knowledge. Some approaches include in their basic definitions some notion of minimality (as the one in Section 7.3), for others specializations of the update definition are suggested, which express minimality requirements and encode them in the update process (cf. [60]). A common approach is to express minimality over rules instead of atoms, establishing a sort of causal dependency among rules.

Rule overriding. Another feature on which further investigation is required is the handling of overridden rules. In some approaches these rules are simply deleted, while in others rules which are rejected by other rules having higher priority are only "deactivated". The rule is still present in the knowledge base but is inhibited (Sections 7.5, 7.7, and [60]). The deactivated rule can also be reactivated in future updates, if some newly added rule can in turn override the rule that caused the initial rejection. A problem which should be addressed, as suggested in [34], is *forgetting* of rules. As far as the knowledge base of a real agent has size constraints, rules cannot be simply added and further deactivated or reactivated infinitely. For space reasons, some rules must sooner or later be physically deleted, and suitable deletion criteria are needed, e.g. the number of updates since the last deactivation without further reactivation, or some utility factor. Observe that this feature is related to the problem of forgetting data in temporal relational databases. Different methods have been suggested for *vacuuming* relational databases, which is non-trivial.

Other improvements or extensions of the formalisms are clearly significant for information agents. An interesting feature is the one suggested in [8], namely the capability of performing updates along several dimensions at the same time. This would be especially interesting to combine it with the issue of nondeterminism in temporal evolution, as indicated in Section 9.

A further interesting issue is the investigation of properties of update mechanisms. The work in [60] is a first attempt in this direction.

10.4 Temporal Reasoning

Temporal reasoning for information agents can be viewed from different perspectives, originating from different requirements respectively. On the one hand, this has been addressed in conjunction with actions and planning [107, 45]. Reasoning tasks involve here the evolution of the worlds, and have to face various aspects, including:

- the action choices available to the agent itself;
- their (possibly nondeterministic [82]) effects;
- the possible results of concurrent actions [23];
- uncertainties about the effective state of the world [81];
- events in the world not known to the agent in advance and possibly affecting its decisions.

A number of *action languages* have been developed and discussed (cf. [78, 107]), as to their expressivity and the possibility of querying knowledge at different stages of the evolution.

On the other hand, LUPS [13], as the first language for specifying update transitions of logic programs, has opened a new interesting field of temporal reasoning, namely reasoning over the evolution of the knowledge base. Reasoning about LUPS specifications needs to consider two kinds of updates: updates coming from the world (i.e. new information about a changing world), and updates driven from the specifications of the agent, defining how they are to be integrated. Both of them can be affected by nondeterminism and uncertainty, whereas the current possibilities of LUPS do not include nondeterministic specifications of updates, and the only uncertainties come from events in the world which are not under the control of the agent. This and other issues need to be better investigated in order to understand what the desiderata for a specification language for logic program updates are, and whether LUPS needs to be extended for some special purposes.

While the first point of view of temporal reasoning often employed the situation calculus, an interesting tool for the second approach could be found in temporal reasoning and in model checking (cf. [46, 45]), choosing a branching-time logic similar to the BDI framework [81]. Advantages could also be drawn from further comparisons between different action languages and LUPS (or other specification languages yet to come).

Another interesting feature is the possibility of reasoning over the past, not only over the future. In BDI logic and in other approaches, several possible futures are considered but only one past exists. While each logic program, in an update sequence, is the result of a precise sequence of successive updates, reasoning over the past could nevertheless be an interesting feature, e.g. when introducing learning functionalities or utility measures over rules, or as a further element in decision making.

11 Conclusion

In this paper, we tried to identify how declarative methods developed in the field of logic programming could be successfully employed for developing advanced reasoning capabilities for information agents. We described the role such agents could have in a multi-agent system for intelligent information access, together with the tasks an information agent or a set of information agents is required to perform.

We shortly reviewed some existing systems and platforms addressing the topic of information integration, mostly based on procedural rather than declarative approaches, and we discussed some interesting ongoing projects as well, in order to have an overview of the most common problems and of some successful solutions as to the implementation of the required features.

As our main focus is on declarative methods in the field of logic programming, we then discussed several declarative approaches for specific reasoning tasks for information agents, namely tasks for preference handling, for revision and update, for expressing quantitative information, and for reasoning over temporal evolution.

We then provided for each of these tasks a tentative evaluation of the declarative formalisms we reviewed, trying to identify which of them are suitable to address which particular problem information agents are faced with. On the basis of this evaluation, some open problems and paths for future research can be now identified, which we briefly discuss in the following.

Decision Making Based on Quantitative Information. The decision making capabilities of information agents could be improved by encoding some form of quantitative information into a logic programming representation, expressing either costs, resource bounds, or uncertainty and probabilities. Many different formalisms and semantics have been suggested in the field.

A first step should consist in a thorough analysis of each of these approaches, or of a class of similar ones, in order to identify for which information problems they are more suitable, in terms of reasoning capabilities for an agent, which requirements they satisfy, and which drawbacks they present. In a second step, the theoretical and practical issues concerning the integration of quantitative information with other reasoning capabilities required for information agents (e.g., revision or update management) should be addressed, investigating possible integrating frameworks for a qualitative and quantitative decision making process. A proposal, though not strictly concerned with logic programming, is outlined in [58].

Declarative Strategies for Information Merging. A promising application domain for testing the decision making capabilities of information agents is the classification and merging of the content of XML documents. This application domain, which is relevant for many Web systems, would profit from the declarative specification of strategies for information merging or classification of semi-structured data, and in particular data stored in XML, which offers promising structural properties. This issue has recently been addressed using techniques based on default logic [86, 87], description logics [42, 43], object oriented formalisms [1], and automated theorem proving and machine learning techniques [137, 138].

The target is to express such strategies declaratively in the knowledge base of the agent, depending on the available knowledge about the particular application domain and the data structure of the available sources, and employing the declarative strategies for preference handling, which are already available in some of the formalisms we discussed.

Unified Framework for Preference and Update Handling. In the light of the work in the field of updates of logic programs, a homogeneous framework could be defined, allowing to handle both updates and preferences in a uniform way. This would be interesting to information agents, as far as it allows to address the problem of updates with information coming from different information sources, together with meta-information such as priority or reliability of the source itself. From this point of view, the agent is faced with several updates, having to consider on the one hand the temporal sequence (since new updates have priority over old information), and on the other hand the relative priorities of the sources or of the information they provide. Work on this issue has already been done in [70, 12, 8], partly based on the approach suggested in [35], but further research is needed.

Temporal Reasoning over Knowledge Base Evolution. An interesting open issue is reasoning about the evolution of the knowledge base, drawing conclusions over sequences of updates. Based on the declarative specifications guiding the evolution, e.g., in form of LUPS programs [13] or some other specification language for logic program updates, whose desired features have to be carefully defined, the target is to perform temporal reasoning over the evolution of the knowledge base, answering questions about the possibility of reaching some desired condition or violating some intended constraint.

These reasoning capabilities naturally rely on temporal logics [64], and could take advantage of techniques and tools developed in the field of model checking [47] or planning [46, 45]. Features allowing nondeterminism in the specifications, e.g. if the evolution depends on external events not known to the agent in advance or the language itself allows nondeterministic updates, could enrich the representation, also taking advantage of similar work in the field of action languages.

The goal of reasoning tasks over the knowledge base and the specifications for its evolution is to know whether the agent behaves correctly, how the epistemic state of the agent changes over time, if uncertain classes of updates (e.g. possibly leading to states violating some constraint) are going to be performed.

References

[1] S. Abiteboul, V. Christophides, S. Cluet, and M. Scholl. From Structured Documents to Novel Query Facilities. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 313–324, 1994.

[2] S. Abiteboul, J. McHugh, D. Quass, J. Widom, and R. Goldman. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997.

[3] S. Abiteboul, J. McHuhg, D. Quass, J. Widom, and J. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[4] S. Adali, A. Brink, R. Emery, J. Lu, A. Rajput, T. Rogers, R. Ross, V.S. Subrahmanian, and C. Ward. HERMES: A Heterogeneous Reasoning and Mediator System, 1993.

[5] S. Adali and R. Emery. A Uniform Framework for Integrating Knowledge in Heterogeneous Knowledge Systems. In A. Chen and P. Yu, editors, *Proceedings of the Eleventh Conference on Data Engineering*, Mar. 1995.

[6] S. Adali and V.S. Subrahmanian. Amalgamating Knowledge Bases, III: Algorithms, Data Structures, and Query Processing. *Journal of Logic Programming*, 28(1):45–88, 1996.

[7] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the Logic of Theory Change: Partial Meet Functions for Contraction and Revision. *Journal of Symbolic Logic*, 50:510–530, 1985.

[8] J. Alferes, J. Leite, and L. Pereira. Multi-dimensional Dynamic Logic Programming. In F. Sadri and K. Satoh, editors, *Proceedings of the CL-2000 Workshop on Computational Logic in Multi-Agent Systems* (*CLIMA 2000*), July 2000.

[9] J. Alferes, J. Leite, L. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic Logic Programming. In A. Cohn, L. Schubert, and S. Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning* (*KR'98*), pages 98–111. Morgan Kaufmann, 1998.

[10] J. Alferes, J. Leite, L. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic Updates of Non-Monotonic Knowledge Bases. *Journal of Logic Programming*, 45(1–3):43–70, 2000.

[11] J. Alferes and L. Pereira. Update-Programs Can Update Programs. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Proceedings of Non-Monotonic Extensions of Logic Programming* (*NMELP'96*), volume 1216 of *Lecture Notes in AI* (*LNAI*), pages 110–131. Springer Verlag, 1997.

[12] J. Alferes and L. Pereira. Updates plus Preferences. In M. Aciego, I. de Guzmán, G. Brewka, and L. Pereira, editors, *Proceedings of the Seventh European Workshop on Logic in Artificial Intelligence* (*JELIA 2000*), volume 1919 in Lecture Notes in AI (LNAI). Springer Verlag, 2000.

[13] J. Alferes, L. Pereira, H. Przymusinska, and T. Przymusinski. LUPS - A Language for Updating Logic Programs. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*), volume 1730 in Lecture Notes in AI (LNAI), pages 162–176. Springer Verlag, Dec. 1999.

[14] J. Alferes, L. Pereira, H. Przymusinska, T. Przymusinski, and P. Quaresma. Preliminary Exploration on Actions as Updates. In M. Meo and M. Vilares-Ferro, editors, *Proceedings of the 1999 Joint Conference on Declarative Programming* (*AGP'99*), Sept. 1999.

[15] A. Analyti and S. Pramanik. Reliable Semantics for Extended Logic Programs with Rule Prioritization. *Journal of Logic and Computation*, pages 303–324, 1995.

[16] Y. Arens. Services and Information Management for Decision Support. In *Proceedings of the Annual AI Systems in Government Conference* (*AISIG'90*), 1990.

[17] Y. Arens, C. Chee, C. Hsu, and C. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *International Journal of Cooperative Information Systems*, 2(2):127–158, 1993.

[18] Y. Arens and C. Knoblock. Planning and Reformulating Queries for Semantically-modeled Multidatabase Systems. In *Proceedings of the First International Conference on Information and Knowledge Management*, 1992.

[19] Y. Arens, C. Knoblock, and W. Shen. Query Reformulation for Dynamic Information Integration. *Journal of Intelligent Information Systems*, 6(2–3):99–130, 1996.

[20] K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross, and V.S. Subrahmanian. IMPACT: A Platform for Collaborating Agents. *IEEE Intelligent Systems*, 14(2):64–72, 1999.

[21] F. Baader and B. Hollunder. Priorities on Defaults with Prerequisite and their Application in Treating Specificity in Terminological Default Logic. *Journal of Automated Reasoning*, 15:41–68, 1995.

[22] F. Bacchus, A. Grove, J. Halpern, and D. Koller. Statistical Foundations for Default Reasoning. In R. Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (*IJCAI'93*), pages 563–569. Morgan Kaufmann, 1993.

[23] C. Baral and M. Gelfond. Reasoning About Effects of Concurrent Actions. *Journal of Logic Programming*, 31(1–3):85–117, 1997.

[24] C. Baral and M. Gelfond. Reasoning Agents in Dynamic Domains. *Workshop on Logic-Based Artificial Intelligence*, June 1999.

[25] C. Baral, S. Kraus and J. Minker. Combining Multiple Knowledge Bases. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):208–220, 1991.

[26] C. Baral, S. Kraus and J. Minker. Combining Knowledge Bases Consisting of First Order Theories. *Computational Intelligence*, 8(1):45–71, 1992.

[27] C. Baral, S. Kraus and J. Minker. Combining Default Logic Databases. *International Journal of Intelligent and Cooperative Information Systems*, 3(3):319–348, 1994.

[28] B. Bargmeyer, J. Fowler, M. Nodine, and B. Perry. Agent-Based Semantic Interoperability in InfoSleuth. *SIGMOD Record*, 28(1):60–67, 1999.

[29] R. Bayardo, B. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Semantic Integration of Information in Open and Dynamic Environments (Experience Paper). In *SIGMOD CONFERENCE 1997*, pages 195–206, 1997.

[30] W. Bohrer, A. Ngu, and M. Nodine. Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth. In *Proceedings of the Fifteenth International Conference on Data Engineering* (*ICDE'99*), pages 358–365. IEEE Computer Society Press, Mar. 1999.

[31] P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, R. Ross, and V.S. Subrahmanian. *Heterogeneous Agent Systems*. MIT Press, 2000.

[32] G. Brewka. Adding Priorities and Specificity to Default Logic. *Proceedings of the 1994 European Workshop on Logic in Artificial Intelligence* (*JELIA'94*), volume 838 in Lecture Notes in AI (LNAI), pages 247–260. Springer Verlag, 1994.

[33] G. Brewka. Well-Founded Semantics for Extended Logic Programs with Dynamic Preferences. *Journal of Artificial Intelligence Research*, 4:19–36, 1996.

[34] G. Brewka. Declarative Representation of Revision Strategies. In *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI 2000)*, 2000.

[35] G. Brewka and T. Eiter. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence*, 109(1–2):297–356, 1999.

[36] G. Brewka and T. Eiter. Prioritizing Default Logic. In S. Hölldobler, editor, *Intellectics and Computational Logic – Papers in Honor of Wolfgang Bibel*, pages 27–45. Kluwer Academic, 2000.

[37] F. Buccafurri, N. Leone and P. Rullo. Stable Models and their Computation for Logic Programming with Inheritance and True Negation. *Journal of Logic Programming*, 27(1):5–43, 1996.

[38] F. Buccafurri, W. Faber, and N. Leone. Disjunctive Logic Programs with Inheritance. Technical Report DBAI-TR-99-30, Institut für Informationssysteme, Technische Universität Wien, Austria, May 1999. To appear in *Theory and Practice of Logic Programming*.

[39] F. Buccafurri, W. Faber, and N. Leone. Disjunctive Logic Programs with Inheritance. In D. D. Schreye, editor, *Proceedings of the Sixteenth International Conference on Logic Programming (ICLP'99)*, pages 79–93. MIT Press, 1999.

[40] F. Buccafurri, N. Leone and P. Rullo. Semantics and Expressiveness of Disjunctive Ordered Logic. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):311–337, 1999.

[41] F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[42] D. Calvanese, G. D. Giacomo, and M. Lenzerini. What can Knowledge Representation do for Semi-Structured Data? In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 1998)*, pages 205–210. AAAI Press / MIT Press, July 1998.

[43] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Representing and Reasoning on XML Documents: A Description Logic Approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.

[44] P. Cannata, M. Huhns, N. Jacobs, T. Ksiezyk, K. Ong, A. Sheth, M. Singh, C. Tomlinson and D. Woelk. The Carnot Heterogeneous Database Project: Implemented Applications. *Distributed and Parallel Databases*, 5(2):207–225, 1997.

[45] A. Cimatti and M. Roveri. Conformant Planning via Model Checking. In *Proceedings of the European Conference on Planning (ECP'99)*, Sept. 1999.

[46] A. Cimatti, M. Roveri, and P. Traverso. Strong Planning in Non-Deterministic Domains via Model Checking. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS'98)*. AAAI Press, June 1998.

[47] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[48] W. Cohen. A Web-Based Information System that Reasons with Structured Collections of Text. In K. Sycara and M. Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 400–407. ACM Press, May 1998.

[49] W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. In L. Haas and A. Tiwary, editors, *Proceedings ACM SIGMOD International Conference on Management of Data* (*SIGMOD 1998*), pages 201–212. ACM Press, June 1998.

[50] K. Decker, K. Sycara, and M. Williamson. Middle-Agents for the Internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (*IJCAI'97*), volume 1, pages 578–583. Morgan Kaufmann, Aug. 1997.

[51] A. Dekhtyar and V.S. Subrahmanian. Hybrid Probabilistic Programs. *Journal of Logic Programming*, 43(3):187–250, 2000.

[52] J. Delgrande, T. Schaub and H. Tompits. A Compilation of Brewka and Eiter's Approach to Prioritization. *Proceedings of the Seventh European Workshop on Logics in Artificial Intelligence* (*JELIA 2000*), pages 376–390, 2000.

[53] J. Delgrande, T. Schaub, and H. Tompits. Logic Programs with Compiled Preference. In W. Horn, editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence* (*ECAI 2000*). IOS Press, Aug. 2000.

[54] P. Dell'Acqua and L. Pereira. Preferring and Updating with Multi-Agents, Draft, 1999.

[55] P. Dell'Acqua and L. Pereira. Updating Agents. In S. Rochefort, F. Sadri, and F. Toni, editors, *Proceedings of the ICLP'99 Workshop Multi-Agent Systems in Logic* (*MASL'99*), 1999.

[56] Y. Dimopoulos and A. Kakas. Information Integration and Computational Logic. *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-Net*, 2000. URL: http://www.compulog.org/net/Forum/Supportdocs.html.

[57] J. Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In A. Fuhrmann and H. Rott, editors, *Logic, Action and Information. Proceedings of the Konstanz Colloquium in Logic and Information* (*LogIn'92*), pages 241–329. DeGruyter, 1995.

[58] J. Dix, M. Nanni, and V.S. Subrahmanian. Probabilistic Agent Programs. *ACM Transactions on Computational Logic*, 1(2), 2000.

[59] J. Dix, G. Pick, and V.S. Subrahmanian. Meta Agent Programs. *Journal of Logic Programming*, 46(1–2):1–60, 2000.

[60] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Considerations on Updates of Logic Programs. In M. Aciego, I. de Guzmán, G. Brewka, and L. Pereira, editors, *Proceedings of the Seventh European Workshop on Logic in Artificial Intelligence* (*JELIA 2000*), volume 1919 of *Lecture Notes in AI* (*LNAI*). Springer Verlag, 2000.

[61] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. *On Updates of Logic Programs by Inheritance*. Technical Report, Technische Universität Wien, Institut für Informationssysteme, Abteilung Wissensbasierte Systeme, 2000.

[62] T. Eiter, G. Pick, and V.S. Subrahmanian. Heterogeneous Active Agents, I: Semantics. *Artificial Intelligence*, 108(1–2):179–255, 1999.

[63] T. Eiter and V.S. Subrahmanian. Heterogeneous Active Agents, II: Algorithms and Complexity. *Artificial Intelligence*, 108(1–2):257–307, 1999.

[64] E. Emerson. Automated Temporal Reasoning about Reactive Systems. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of Lecture Notes in Computer Science (LNCS), pages 41–101. Springer Verlag, 1996.

[65] E. Emerson and J. Srinivasan. Branching Time Temporal Logic. In J. D. Bakker, W. D. Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop* (*REX Workshop'88*), volume 354 of *Lecture Notes in Computer Science* (*LNCS*), pages 123–172. Springer Verlag, 1988.

[66] P. Flach and A. Kakas, editors. *Abduction and Induction — Essays on Their Relation and Integration*. Kluwer Academic, 2000.

[67] R. Flores-Mendez. Towards a Standardization of Multi-Agent System Frameworks. *ACM Crossroads*, Issue 5.4 on Intelligent Agents, 1999.

[68] N. Foo and Y. Zhang. Answer Sets for Prioritized Logic Programs. In J. Maluszynski, editor, *Proceedings of the 1997 International Logic Programming Symposium*, pages 69–83. MIT Press, Oct. 1997.

[69] N. Foo and Y. Zhang. Towards Generalized Rule-based Updates. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (*IJCAI'97*), volume 1, pages 82–88. Morgan Kaufmann, Aug. 1997.

[70] N. Foo and Y. Zhang. Updating Logic Programs. In H. Prade, editor, *Proceedings of the Thirteenth European Conference on Artificial Intelligence* (*ECAI'98*), pages 403–407. John Wiley and Sons, Aug. 1998.

[71] P. Froehlich, W. Nejdl, M. Schroeder, C. Damásio, and L. Pereira Using Extended Logic Programming for Alarm-Correlation in Cellular Phone Networks. In *Proceedings of the Twelfth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems* (*IEA/AIE'99*). Springer Verlag, 1999.

[72] D. Gabbay, E. Laenens and D. Vermeir. Credulous vs. Sceptical Semantics for Ordered Logic Programs. *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (*KR'92*), pages 208–217, 1992.

[73] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In *Proceedings of the AAAI Symposium on Information Gathering*, pages 61–64, Mar. 1995.

[74] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[75] P. Gärdenfors and H. Rott. Belief Revision. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of logic in Artificial Intelligence and Logic Programming, Volume IV: Epistemic and Temporal Reasoning*, pages 35–132. Oxford Science Publications, 1995.

[76] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080. MIT Press, Aug. 1988.

[77] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3–4):365–386, 1991.

[78] M. Gelfond and V. Lifschitz. Action Languages. *Electronic Transactions on AI*, 3(16), 1998.

[79] M. Gelfond and T. Son. Reasoning with Prioritized Defaults. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Logic Programming and Knowledge Representation, Third International Workshop, Selected Papers* (*LPKR'97*), volume 1471 of *Lecture Notes in Computer Science* (*LNCS*), pages 164–223. Springer Verlag, Oct. 1998.

[80] M. Genesereth. Knowledge Interchange Format. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning* (*KR'91*), pages 599–600. Morgan Kaufmann, Apr. 1991.

[81] M. Georgeff and A. Rao. Modelling Rational Agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Knowledge Representation and Reasoning* (*KR'91*), pages 473–484. Morgan Kaufmann, Apr. 1991.

[82] E. Giunchiglia, G. Kartha, and V. Lifschitz. Representing Action: Indeterminacy and Ramifications. *Artificial Intelligence*, 95(2):409–438, 1997.

[83] E. Giunchiglia and V. Lifschitz. Action Languages, Temporal Action Logics and the Situation Calculus. In *Working Notes of the IJCAI'99 Workshop on Nonmonotonic Reasoning, Action, and Change*, 1999.

[84] M. Huhns and M. Singh. The Semantic Integration of Information Models. In *AAAI Workshop on Cooperation among Heterogeneous Intelligent Agents*, July 1992.

[85] M. Huhns and M. Singh. Automating Workflows for Service Provisioning: Integrating AI and Database Technologies. *IEEE Expert*, 9(5), Oct. 1994.

[86] A. Hunter. Merging Potentially Inconsistent Items of Structured Text. *IEEE Transactions on Knowledge and Data Engineering*, 34(3):305–332, 2000.

[87] A. Hunter. Reasoning with Inconsistency in Structured Text. *Knowledge Engineering Review*, 15(2):1–21, 2000.

[88] K. Inoue and C. Sakama. Abductive Framework for Nonomonotonic Theory Change. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (*IJCAI'95*), pages 204–210. Morgan Kaufmann, Aug. 1995.

[89] K. Inoue and C. Sakama. Representing Priorities in Logic Programs. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming* (*JICSLP'96*). MIT Press, Sept. 1996.

[90] K. Inoue and C. Sakama. Updating Extended Logic Programs through Abduction. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*), volume 1730 in Lecture Notes in AI (LNAI), pages 147–161. Springer Verlag, Dec. 1999.

[91] A. Kakas, R. Kowalski, and F. Toni. The Role of Abduction in Logic Programming. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume V: Logic Programming*, pages 235–324. Clarendon Press, Oxford, 1998.

[92] H. Katsuno and A. Mendelzon. On the Difference between Updating a Knowledge Base and Revising It. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning* (*KR'91*), pages 387–394. Morgan Kaufmann, Apr. 1991.

[93] M. Kifer and V.S. Subrahmanian. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming*, 12(3–4):335–367, 1992.

[94] C. Knoblock. Planning, Executing, Sensing and Replanning for Information Gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (*IJCAI'95*), pages 1686–1693. Morgan Kaufmann, Aug. 1995.

[95] R. Kowalski and F. Sadri. Logic Programs with Exceptions. *New Generation Computing*, 9:387–400, 1991.

[96] R. Kowalski and F. Sadri. Towards a Unified Agent Architecture that Combines Rationality with Reactivity. In D. Pedreschi and C. Zaniolo, editors, *Logic in Databases, International Workshop* (*LID'96*), volume 1154 of *Lecture Notes in Computer Science* (*LNCS*), pages 137–149. Springer Verlag, July 1996.

[97] R. Kowalski and F. Sadri. From Logic Programming towards Multi-agent Systems. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):391–419, 1999.

[98] E. Laenens and D. Vermeir. A Fixpoint Semantics for Ordered Logic. *Journal of Logic and Computation*, 1:159–185, 1990.

[99] E. Lamma, F. Riguzzi, and L. Pereira. Strategies in Combined Learning via Logic Programs. *Machine Learning*, 38(1–2):63–87, 2000.

[100] J. Leite and L. Pereira. Generalizing Updates: from Models to Programs. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Logic Programming and Knowledge Representation, Selected Papers of the Third International Workshop* (*LPKR'97*), volume 1471 of *Lecture Notes in AI* (*LNAI*), pages 224–246. Springer Verlag, Oct. 1997.

[101] A. Levy, A. Rajaraman, and J. Ordille. Query-Answering Algorithms for Information Agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eight Innovative Applications of Artificial Intelligence Conference* (*AAAI/IAAI'96*), pages 40–47. AAAI Press / MIT Press, Aug. 1996.

[102] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In T. Vijayaraman, A. Buchmann, C. Mohan, and N. Sarda, editors, *Proceedings of the 22th International Conference on Very Large Data Bases* (*VLDB'96*), pages 251–262. Morgan Kaufmann, Sept. 1996.

[103] A. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *Journal of Intelligent Information Systems*, 5(2):121–143, 1995.

[104] A. Levy and M. Rousset. CARIN: A Representation Language Combining Horn Rules and Description Logics. In W. Wahlster, editor, *Proceedings of the 12th European Conference on Artificial Intelligence* (*ECAI'96*), pages 323–327. John Wiley and Sons, Aug. 1996.

[105] A. Levy and D. Weld. Intelligent Internet systems. *Artificial Intelligence*, 118(1–2):1–14, 2000.

[106] P. Liberatore and M. Schaerf. Arbitration (or How to Merge Knowledge Bases). *IEEE Transactions on Knowledge and Data Engineering*, 10(1):76–90, 1998.

[107] V. Lifschitz. Action Languages, Answer Sets and Planning. In K. Apt, V. W. Marek, M. Truszczyński, and D. S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.

[108] V. Lifschitz and T. Woo. Answer Sets in General Nonmonotonic Reasoning (Preliminary Report). In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning* (*KR'92*), pages 603–614. Morgan Kaufmann, Oct. 1992.

[109] T. Lukasiewicz. Probabilistic Logic Programming. In H. Prade, editor, *Proceedings of the Thirteenth European Conference on Artificial Intelligence* (*ECAI'98*), pages 388–392. John Wiley & Sons, Aug. 1998.

[110] T. Lukasiewicz. Probabilistic Logic Programming with Conditional Constraints. *ACM Transactions on Computational Logic*, 2001. To appear. Technical Report INFSYS-TR 1843-00-01, Institute of Information Systems, TU Vienna, 2000.

[111] R. MacGregor. Inside the LOOM Description Classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[112] V. Marek and M. Truszczyński. Revision Specifications by Means of Programs. In C. MacNish, D. Pearce, and L. Pereira, editors, *Proceedings of the European Workshop on Logics in Artificial Intelligence* (*JELIA'94*), volume 838 of *Lecture Notes in AI* (*LNAI*), pages 122–136. Springer Verlag, Sept. 1994.

[113] V. Marek and M. Truszczyński. Logic Programming with Costs. Unpublished note, 1999.

[114] W. Marek and M. Truszczyński. *Nonmonotonic Logics – Context-Dependent Reasoning*. Springer Verlag, 1993.

[115] R. Moore. Semantical Considerations on Nonmonotonic Logics. *Artificial Intelligence*, 25(1):75–94, 1985.

[116] B. Nebel. How Hard is it to Revise a Belief Base ? In D. Dubois and H. Prade, editors, *Handbook on Defeasible Reasoning and Uncertainty Management Systems, Volume III: Belief Change*, pages 77–145. Kluwer Academic, 1998.

[117] R. Ng. Semantics, Consistency, and Query Processing of Empirical Deductive Databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(1):32–49, 1997.

[118] R. Ng and V.S. Subrahmanian. Probabilistic Logic Programming. *Information and Computation*, 101(2):150–201, 1992.

[119] R. Ng and V.S. Subrahmanian. A Semantical Framework for Supporting Subjective and Conditional Probabilities in Deductive Databases. *Journal of Automated Reasoning*, 10(2):191–235, 1993.

[120] R. Ng and V.S. Subrahmanian. Stable Semantics for Probabilistic Deductive Databases. *Information and Computation*, 110(1):42–83, 1994.

[121] I. Niemelä, P. Simons, and T. Soininen. Stable Model Semantics of Weight Constraint Rules. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*), volume 1730 of *Lecture Notes in Computer Science* (*LNCS*), pages 317–331. Springer Verlag, Dec. 1999.

[122] D. Nute. Defeasible Logic. In D. Gabbay, C. Hogger and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume III*, pages 353–395. Clarendon Press, 1994.

[123] J. Pearl. Probabilistic Semantics for Nonmonotonic Reasoning: A Survey. In R. Brachman, H. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* (*KR'89*), pages 505–516. Morgan Kaufmann, May 1989.

[124] D. Poole. Logic Programming, Abduction and Probability. *New Generation Computing*, 11(3):377–400, 1993.

[125] S. Pradhan and J. Minker. Using Priorities to Combine Knowledge Bases. *International Journal of Cooperative Intelligent Systems*, 5(2–3):333–364, 1996.

[126] H. Prakken and G. Sartor. Argument-based Extended Logic Programming with Defeasible Priorities. *Journal of Applied Non-Classical Logics*, 7:25–75, 1997.

[127] T. Przymusinski and H. Turner. Update by Means of Inference Rules. *Journal of Logic Programming*, 30(2):125–143, 1997.

[128] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying Semistructured Heterogeneous Information. In T. Ling, A. Mendelzon, and L. Vieille, editors, *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases* (*DOOD'95*), volume 1013 of *Lecture Notes in Computer Science* (*LNCS*), pages 319–344. Springer Verlag, Dec. 1995.

[129] I. Ramakrishnan, P. Rao, K. Sagonas, T. Swift and D. Warren. Efficient Tabling Mechanisms for Logic Programs. In L. Sterling, editor, *Proceedings of the Twelfth International Conference on Logic Programming* (*ICLP'95*), pages 697–711. MIT Press, Jun. 1995.

[130] P. Revesz. On the Semantics of Theory Change: Arbitration between Old and New Information. *Proceedings of the Twelfth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems* (*PODS'93*), pages 71–79, 1993.

[131] J. Rintanen. Lexicographic Priorities in Default Logic. *Artificial Intelligence*, 106:221–265, 1998.

[132] F. Sadri and F. Toni. Computational Logic and Multi-Agent Systems: a Roadmap *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-Net*, 2000. URL: http://www.compulog.org/net/Forum/Supportdocs.html.

[133] K. Sagonas, T. Swift, and D. Warren. XSB as an Efficient Deductive Database Engine. In R. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 442–453. ACM Press, May 1994.

[134] C. Sakama. Abductive Generalization and Specialization. In P. Flach and A. Kakas, editors, *Abduction and Induction - Essays on their Relation and Integration*, pages 253–265. Kluwer Academic, 2000.

[135] V.S. Subrahmanian. On the Semantics of Quantitative Logic Programs. In *Proceedings of the Fourth IEEE Symposium on Logic Programming* (*SLP'87*), pages 173–182. Computer Society Press, 1987.

[136] V.S. Subrahmanian. Amalgamating Knowledge Bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.

[137] B. Thomas. Intelligent Web Querying with Logic Programs. In *Proceedings of the Workshop on Inference Mechanisms in Knowledge-based Systems at the National German AI Conference* (*KI'98*), 1998.

[138] B. Thomas. Logic Programs for Intelligent Web Search. In Z. Ras and A. Skowron, editors, *Proceedings of the Eleventh International Symposium on Foundations of Intelligent Systems* (*ISMIS'99*), volume 1609 of *Lecture Notes in Computer Science* (*LNCS*). Springer Verlag, June 1999.

[139] M. H. van Emden. Quantitative Deduction and its Fixpoint Theory. *Journal of Logic Programming*, 3(1):37–53, 1986.

[140] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, Mar. 1992.