TLP **2** (6): 645–709, November 2002. © 2002 Cambridge University Press
DOI: 10.1017/S1471068402001576 Printed in the United Kingdom

Using methods of declarative logic programming for intelligent information agents

THOMAS EITER, MICHAEL FINK,

GIULIANA SABBATINI and HANS TOMPITS

Institut für Informationssysteme, Technische Universität Wien,
Abt. Wissensbasierte Systeme 184/3, Favoritenstrasse 9-11, A-1040 Vienna, Austria
(*e-mail:* [eiter,michael,giuliana,tompits]@kr.tuwien.ac.at)

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 requires advanced capabilities to address complex tasks, 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 lies in the role which some methods from the field of declarative 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 specific needs of this application domain. 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.

KEYWORDS: multi-agent systems, logic programming, information agents

1 Introduction

Currently, 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, 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 an appropriate way, depending on the interfaces available, as well as on the data organization and presentation. Furthermore, the user must possibly adapt the

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

query more than once for each source to get the desired information, and must learn several different query approaches. Further, 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 multi-agent systems 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 requires 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 'pure' logic programming, and in particular answer set semantics and its derivatives, can play in the realization of reasoning capabilities for information agents. These methods lack richer algorithmic control structures as those typical of Prolog, and permit knowledge representation in a purely declarative way, without the need for paying much attention to syntactical matters such as the way in which rules are listed in a program or literals occur in the body of a rule. This seems to be well-suited in the context of rule-based decision making, where semantical aspects for a 'rational' decision component are the main concern rather than the control structure. In fact, the rational decision component will be embedded into some control structure, which calls the component.

The main focus of this paper is to discuss how the methods from pure (or 'declarative') logic programming can be used, extended, and further developed for

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

the specific needs of information agents. As we shall see, the methods which are available to date do not meet all the needs which are required, which leaves the way open for interesting research.

The paper is organized as follows. Section 2 gives a detailed description of the role of information agents in the field of advanced information access, and discusses the architecture of a prototypical agent system. In section 3, we identify a set of reasoning capabilities which are mandatory for an advanced information-integration system, some of which are critical for an 'intelligent' information agent. Section 4 contains a brief review of some existing systems and current projects which address typical information integration problems, most of them by means of ad hoc procedural techniques. Although we are more interested in declarative methods, the different approaches implemented in these systems provide relevant information concerning possible solutions and hint to intrinsic difficulties.

We then focus on declarative knowledge representation methods. First, section 5 discusses some subtasks amenable for information agents, which promise to have a successful solution based on declarative methods. In sections 6–9, we then review some approaches and methodologies from 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, we evaluate the presented methods, taking into account their applicability in the field of information agents. We discuss advantages and drawbacks, and address the possibility of extensions.

Section 11 concludes the paper, containing some open issues which emerge from this analysis, and outlines directions for future research.

2 Intelligent information agents

Several models of multi-agent systems 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 focus on *knowledge-based approaches* for developing reasoning components for intelligent information access (for an extensive overview of agent programming approaches, cf. Sadri and Toni, 2000). In particular, we are interested in *information agents* (sometimes also called *middle agents* (Decker *et al.*, 1997)) 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 Flores-Mendez (1999), 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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

user requests. Brokers receive requests, look for relevant sources matching these requests, and then perform actions using services from other agents (combining them with their own resources or information).

Mediators: in the mediator approach (Wiederhold, 1992), meta-knowledge about a number of other agents (sometimes called *provider agents*) is available to the mediator, which exploits this knowledge 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, an 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 focus our attention on information agents which act as mediators; in particular, we are interested in the *rational capabilities* of such agents. For this purpose, we assume that the agent has knowledge about the application domain of interest, together with some meta-knowledge about the contents and features of the distributed, heterogeneous information sources the system has access to. This knowledge is represented in the form of declarative rules, determining the beliefs and actions of the agent. For example, in an application where bibliographic references should be found, the agent must have some knowledge about the structure of references, the information fields which are necessary to identify a work, and so on, together with further knowledge about the information sources which it might consult, such as their reliability and availability, what kind of works they offer, and others.

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

- identifying and possibly ranking relevant information sources (e.g., selection of Web pages);
- 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 (i.e., if more than one source has been queried);
- optimizing the number of accessed sources, or the total cost and time required for the search, or the precision and completeness of the results.

For executing these tasks, the agent has to make decisions about what actions should be performed or how to proceed at certain points. An 'intelligent' agent makes these decisions based on reasoning methods using its knowledge about the

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

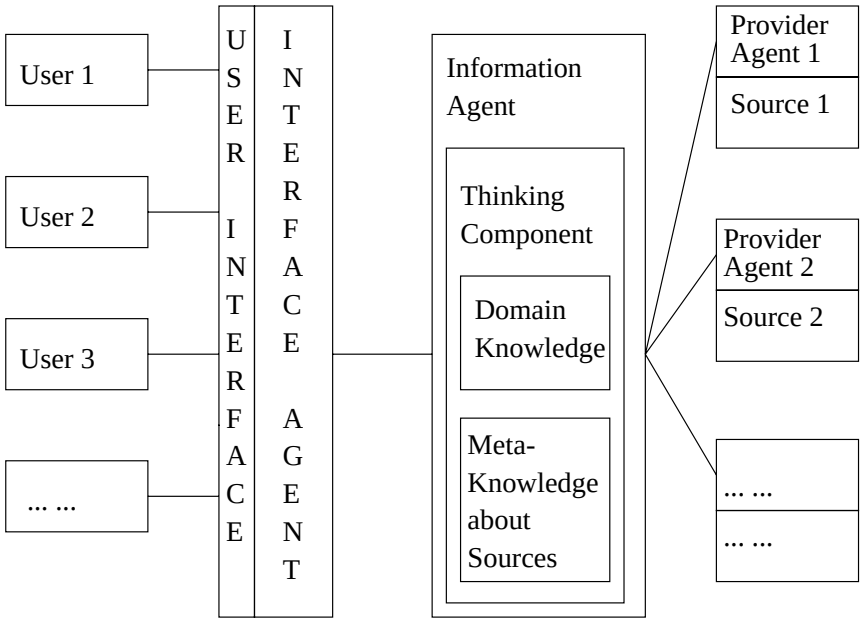


Fig. 1. System architecture.

application domain and the particular requests 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 procedure 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. Here, we restrict our attention to the use of non-Prolog logic programming (mostly based on the answer set semantics) for this purpose. It is important to point out, however, that we suggest the use of declarative logic programming *for realizing some rational components of the agent*, whereas other programming languages and environments may be more suitable for implementing other components, like control and communication (infra-)structures (cf. section 4.11.2).

As for an agent-based information system, different architectures can be envisaged, in which information agents have different capabilities and roles. In this paper, we assume the following prototypical system architecture, as illustrated in Figure 1.

- We have a multi-agent system providing access to several heterogeneous information sources. In such a system, the existence of an *interface agent* is assumed, which offers an homogeneous user interface, collects user requests (typically from the Web), and formulates them in a language suitable for subsequent processing. 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, e.g., KQML (Finin *et al.*, 1994) –

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 and passes it to the interface agent, which in turn informs the user in a suitable way.

Information sources and the 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, 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 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 fits into is IMPACT, the Interactive Maryland Platform for Agents Collaborating Together (Subrahmanian *et al.*, 2000), which we briefly discuss in section 4.8. 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 somehow be realized in the system; 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, whose features 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. For example, if the user looks for a restaurant and wants to see where it is located, the query can be decomposed into two parts: (i) search for a restaurant satisfying certain criteria; and (ii) search 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 (Levy *et al.*, 1996a; Levy *et al.*, 1996b; Levy *et al.*, 1995; Cannata *et al.*, 1997; Huhns and Singh, 1992; Bargmeyer *et al.*, 1999; Bayardo *et al.*, 1997; Bohrer *et al.*, 1999; Arens *et al.*, 1993; Arens *et al.*, 1996; Garcia-Molina *et al.*,

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

1995; Garcia-Molina *et al.*, 1997; Quass *et al.*, 1995; Abiteboul *et al.*, 1997a; Abiteboul *et al.*, 1997b; Adali *et al.*, 1995; Adali and Emery, 1995).

- **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. For example, if a user is looking for Web pages about a given topic, and the system knows that the user does not speak English, then pages containing text in his/her native language should be preferred. This is useful not only for human users but for servicing other software as well. For instance, 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 (Levy *et al.*, 1996a; Levy *et al.*, 1996b; Levy *et al.*, 1995; Cannata *et al.*, 1997; Huhns and Singh, 1992; Bargmeyer *et al.*, 1999; Bayardo *et al.*, 1997; Bohrer *et al.*, 1999; Adali *et al.*, 1995; Adali and Emery, 1995; Arens *et al.*, 1993; Arens *et al.*, 1996; Garcia-Molina *et al.*, 1995; Garcia-Molina *et al.*, 1997; Quass *et al.*, 1995). If possible, determine further preferred information sources to be queried.
- **Create a query plan.** Determine in which order sub-queries are to be performed (Levy *et al.*, 1996a; Levy *et al.*, 1996b; Levy *et al.*, 1995; Arens *et al.*, 1993; Arens *et al.*, 1996), on the basis of query decomposition, of the data available in every source, and of preference over sources, in order to optimize the execution. For example, 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 (Levy *et al.*, 1996a; Levy *et al.*, 1996b; Arens *et al.*, 1993; Arens *et al.*, 1996; Knoblock, 1995).
- **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 (Cannata *et al.*, 1997; Huhns and Singh, 1992; Bargmeyer *et al.*, 1999; Bayardo *et al.*, 1997; Adali *et al.*, 1995; Adali and Emery, 1995; Garcia-Molina *et al.*, 1995; Garcia-Molina *et al.*, 1997; Quass *et al.*, 1995).
- **Detect and possibly remove inconsistencies** among retrieved data (Cannata *et al.*, 1997; Huhns and Singh, 1992; Adali *et al.*, 1995; Adali and Emery, 1995) on the basis of an inconsistency removal strategy and of meta-knowledge about the sources, or meta-knowledge contained in the answers themselves. For

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

example, some source may return data together with a reliability or probability value for those data.

- **Integrate incomplete information** by means of internal reasoning capabilities (Garcia-Molina *et al.*, 1995; Garcia-Molina *et al.*, 1997; Quass *et al.*, 1995). For example, if in a geographical database there is no information available about a specific address, but the address is located in a certain district or near a known place, it could be useful to search for a map of the corresponding district, giving the user at least an approximative information.
- **Start a learning procedure** to improve or update internal models and reasoning rules (Arens *et al.*, 1993; Arens *et al.*, 1996), perhaps providing input to the profiling agent as well. For example, if a certain source is classified as providing an answer in one second, and it required for each of the last ten queries 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 by Levy and Weld (2000), 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 recommender 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 Levy and Weld (2000).
- **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 outlined 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 these approaches and the proposed solutions are of interest for the whole field of heterogeneous information integration. In the following sections, we consider how declarative formalisms (mostly based on declarative logic programming under the answer set semantics) can help in improving some rational capabilities of information

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

agents. For a comprehensive overview of existing systems in the field of information integration and of the role of logic programming in some of them (cf. Dimopoulos and Kakas, 2000).

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, or languages.

4.1 Cohen's information system for structured collections of text

The system of Cohen (1998b) processes movie review data extracted from Web information sources. Text sources are converted into a highly structured collection of 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 (Cohen, 1998a), 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.

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

4.2 Information manifold

The Information Manifold (Levy *et al*., 1996a, 1996b, 1995) 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 (1996) and 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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 reorganized. 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 (cf. Levy *et al.*, 1996a, 1996b). The aim of the planning tool is the minimization of the number of information sources to be accessed.

4.3 Carnot

Carnot (Huhns and Singh, 1992; Cannata *et al.*, 1997) 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 ontologies, which describe the semantical structure 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 applications (Huhns and Singh, 1994), data validation and cleaning of large databases, and information retrieval from structured databases in response to text queries through model integration of heterogeneous resources.

4.4 InfoSleuth

InfoSleuth (Bargmeyer *et al.*, 1999; Bayardo *et al.*, 1997; Bohrer *et al.*, 1999) is a Web agent system based on the results of the Carnot project. It uses domain ontologies which provide a declarative description of the semantic information inherent to a specific domain, independently of 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. User requests can be specified independent of structure, location, and existence of data, and are then matched with the relevant information sources at

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

processing. This allows the matching procedure also to take into account 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 (Genesereth, 1991), 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 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 applied treatments, and in the environmental domain as a component of EDEN (Environmental Data Exchange Network).

4.5 Infomaster

Infomaster (Genesereth *et al*., 1997) is a system for information integration which provides the user with homogeneous access to multiple distributed heterogeneous information sources. These may be either databases, which are accessed by means of wrappers, or structured Web sources.

In Infomaster, a facilitator dynamically determines, for each user request, the most efficient query plan, trying to minimize the number of accessed sources and coping with their heterogeneity, both in structure and content. An overview of query planning capabilities in Infomaster can be found in Duschka and Genesereth (1997). Infomaster has been used at Stanford University, and is now commercially available.

4.6 COIN

COIN (COntext INterchange) (Bressan *et al*., 1997; Bressan and Goh, 1998) is an information integration framework addressing semantic heterogeneity of data and possible inconsistencies arising from different semantic interpretations of information. It is based on a shared application domain model and on declarative definitions of how different pieces of information have to be integrated. Utilizing a uniform internal data model and language, the so-called *context mediator* rewrites and provides answers to user queries, which do not need to be expressed in the language of the respective information source. Query optimization features are also provided by the system.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

4.7 HERMES

The Heterogeneous Reasoning and Mediator System (HERMES) (Adali *et al*., 1995; Adali and Emery, 1995; Adali and Subrahmanian, 1996; Subrahmanian, 1994) is based on a declarative language for the definition of a mediator (Wiederhold, 1992), 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 how 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 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.

4.8 IMPACT

In the Interactive Maryland Platform for Agents Collaborating Together (IMPACT) (Arisha *et al*., 1999; Subrahmanian *et al*., 2000; Eiter *et al*., 1999; Eiter and Subrahmanian, 1999), 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 which the agents require in order to cooperate, including yellow pages, a thesaurus and registration facilities.

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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 U.S. Army. Other applications have been developed for department stores, for solving the *controlled flight into terrain* problem, and for supply chain management.

4.9 SIMS

The Single Interface to Multiple Sources effort (SIMS) (Arens *et al*., 1993, 1996) 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 searching, as well as for learning.

The system can be combined with Oracle databases as well as with LOOM (MacGregor, 1991) 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 (Arens and Knoblock, 1992), to the control of the status of the U.S. Pacific Fleet (Arens, 1990), and to decision support systems for medical trauma care.

4.10 TSIMMIS

The Stanford-IBM Manager of Multiple Information Sources (TSIMMIS) (Garcia-Molina *et al*., 1995, 1997; Quass *et al*., 1995) integrates heterogeneous information sources including structured and unstructured data, to be used in decision support systems.

The architecture of the system includes several components:

1. a **classifier/extractor** for the extraction of properties from unstructured objects, based on the RUFUS System (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**.

LOREL (Quass *et al*., 1995; Abiteboul *et al*., 1997b; Abiteboul *et al*., 1997a) is the specially developed query language 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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

- answered meaningfully, even when some data is missing, by means of partial type/object assignments;
- 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;
- implemented flexibly, to deal with the possibility of a query resulting in heterogeneous sets; and
- executable even if the object structure is not fully known, by means of wild-cards over attribute labels.

4.11 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.

4.11.1 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 (Dell'Acqua and Pereira, 1999b) is built upon the XSB-Prolog system (Ramakrishnan *et al.*, 1995; Sagonas *et al.*, 1994) and combines KS-Agents (Kowalski and Sadri, 1996), the dynamic logic programming paradigm for updates (cf. section 7.5), and tabling and abduction.

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, updates of logic programs (Alferes *et al.*, 1998; Alferes *et al.*, 2000a; Alferes *et al.*, 1999a), belief revision, and abduction. A prototype for alarm-correlation in cellular-phone networks has been implemented (Fröhlich *et al.*, 1999).

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

- actions and observations (Alferes *et al.*, 1999b);
- coordination of interaction protocols;
- individual and joint planning of actions;
- non-interference behavior;

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

- resource sharing;
- self and mutual diagnosis of imperfect knowledge.

Learning, addressing the following issues (Lamma *et al.*, 2000):

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

4.11.2 Logic programming for multi-agent systems

The role of logic programming in multi-agent systems is usually 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 suggested by Kowalski and Sadri (1999), it is nevertheless possible to extend logic programming techniques for providing 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 the knowledge bases of multiple agents into account, 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 (Georgeff and Rao, 1991) 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, constraints, actions and commands.

Interesting applications of this extension of logic programming would be in the field of distributed information integration, and in the field of network management.

4.11.3 MIA: Mobile Information Agents

The goal of this project is to develop 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. For instance, a user arrives late in the night in some foreign 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 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 knowledge present in the system, with respect to the user location. The user is directly presented with the required information, preprocessed and merged.

The agents building up the knowledge of the system are autonomous in detecting

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Web pages which are of interest with respect to some specific topic, and in extracting relevant information, based on its structure and content. The representation and reasoning components are based on logic programming and automated theorem proving methods, together with machine learning techniques. For more details about the project, see Thomas (1998, 1999).

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 the form of representation and reasoning capabilities which the information agent (or some module) must have. For some of these reasoning tasks, listed below, methods of declarative logic programming (mainly based on the answer set semantics) seem to be promising, and more complex control structures, such as in Prolog, are not necessarily required. An advantage of these formalisms is their well-defined semantics, which supports a strict formalization of reasoning procedures and their outcomes.

Priority handling. Priorities may result from information the agent possesses about reliability or accuracy of knowledge pieces or knowledge sources. Priority information makes it possible, for example, to select the appropriate sources, to merge answers from different sources, and to remove possible inconsistencies. Dealing with priority in the logic programming context has received considerable attention (Gelfond and Son, 1998; Brewka and Eiter, 1999; Inoue and Sakama, 1996; Foo and Zhang, 1997a; Delgrande *et al.*, 2000c; Brewka, 2000; Buccafurri *et al.*, 2001; Buccafurri *et al.*, 1999a; Alferes and Pereira, 2000). Priority information needs to be encoded in the knowledge base of the agent, possibly in the object language itself in which the knowledge 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 (e.g., the application domain itself, other agents inside or outside the system, or sensing actions), or from internal learning procedures. When incorporating this new knowledge into the current knowledge base, conflicts with available information may arise, and revision or update strategies are required to yield a consistent knowledge base (Brewka, 2000; Marek and Truszczyński, 1994; Leite and Pereira, 1997; Inoue and Sakama, 1995; Sakama, 2000; Inoue and Sakama, 1999; Foo and Zhang, 1997b; Foo and Zhang, 1998; Alferes *et al.*, 1998; Alferes *et al.*, 2000a; Alferes and Pereira, 1997; Alferes and Pereira, 2000; Alferes *et al.*, 1999a). The epistemic state and the intended belief set of the agent have thus to be continuously revised, and the revision or update policy governing this process could in turn be explicitly described in the knowledge base and may be dynamically updated too.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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. Strategies for inconsistency removal may be based on explicit or implicit preference information, e.g., the application domain may require old information to be preserved against new one, thus old knowledge is to be preferred to solve inconsistencies.

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. We are not addressing this task in this paper.

Temporal reasoning. The evolution of a dynamic knowledge base could be subject to external specifications (Alferes *et al*., 1999a), describing the content of a sequence of update programs under dynamic conditions, or to internal update policies, defining the way in which the knowledge base is going to evolve for some specific situation. Corresponding forms of reasoning about knowledge evolution have to be provided in order to ensure that the agent's behavior is appropriate, for instance, that some undesired status cannot be reached (Emerson, 1996; Clarke *et al*., 2000), or that the agent may arrive at some goal state.

Learning. Based on the history of the agent (like a sequence of changes in its knowledge base, or a sequence of observations), some form of inductive learning could be implemented (Sakama, 2000; Inoue and Sakama, 1999; Dell'Acqua and Pereira, 1999b). This topic is also not addressed in this paper.

In the following, we deal with formalisms 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 (cf. also section 10), and we usually provide examples illustrating the different approaches.

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, which have been extensively studied in the literature and proven to be useful, such as:

- abduction and abductive logic programming (Kakas *et al*., 1992; Kakas *et al*., 1998; Sakama, 2000); the relationship between abductive and disjunctive resp. normal logic programming (Inoue and Sakama, 1994; Inoue and Sakama, 2000a; Kowalski and Toni, 1995); learning with abduction (Kakas and Riguzzi, 1997); cf. also references in section 7.3;
- induction and abduction (Lamma *et al*., 1999; Flach and Kakas, 2000); cf. also (Sakama, 2000);
- argumentation (Dung, 1993; Bondarenko *et al*., 1997; Kowalski and Toni, 1996); argumentation for reasoning about actions and change (Kakas *et al*.,

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

1999); argumentation and priorities (Prakken and Sartor, 1997; Prakken and Sartor, 1999); argumentation and negotiation among agents (Jennings *et al.*, 1997).

A broad overview with further references can be found in Sadri and Toni (2000).

5.2 Short review of logic programs and answer sets

We recall some notions of logic programming and the answer set semantics, which shall be useful in the following sections. We will deliberatively not focus on all technical details, however.

A logic program consists of a (finite) set of rules $Head \leftarrow Body$, where *Head* is a literal and *Body* is a (possibly empty) conjunction of literals. Rules with empty *Body* are called *facts*. Literals are built over a set of atoms using default negation *not* (also referred to as *negation as failure*, or *weak negation*), and strong negation $\neg$ (also called *classical negation*).

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

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

Informally, rule r_1 says that a site is up by default, and r_2 expresses that if a site is down, then it cannot be queried. Rule r_3 states that the science citation index site (*index*) is queried by default, providing the site is up, while rule r_4 states that, by default, the *cite_seer* site is queried if *index* is down but *cite_seer* is up. Rule r_5 says that if both *index* and *cite_seer* are down, then an error is flagged.

If, as in this example, default negation does not occur in the heads of rules, then 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, allowing rules with empty heads (*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.

There exist several semantics of ELPs (for an overview, cf. Dix, 1996). One of the most widely used is the *answer set semantics* (Gelfond and Lifschitz, 1991), which generalizes the stable model semantics for NLPs (Gelfond and Lifschitz, 1988).

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Similar definitions for GLPs and other classes of programs can be found in the literature (e.g., see Lifschitz and Woo, 1992).

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. For a classical literal L, where $L = A$ or $L = \neg A$ for some atom A, we say that L is *true in* I if $L \in I$, while *not* L is true in I if $L \notin I$. An interpretation I satisfies a ground rule r, denoted by $I \models r$, 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,* P^I*, of* P *with respect to* 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 bodies of 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.

Example 2
Consider the program P 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 may possess none, one, or several 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:

$$\begin{aligned} J_1 \ = \ \{&\neg up(index), \neg rel(cite_seer), up(cite_seer), \\ &\neg query(index), query(cite_seer)\}, \end{aligned}$$

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

$$J_2 = \{\neg up(index),\ \neg rel(cite_seer),\ up(cite_seer),\ \neg query(index),\ \neg query(cite_seer)\}.$$

The set of consequences of a 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 Example 3, multiple answer sets exist, preferences might be used to single out either a unique 'preferred' answer set, or a collection of preferred ones. 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. Different ways to represent and handle preferences on logic programs are reviewed in the next section.

Concerning updates, suppose a new fact $\neg up(index) \leftarrow$ should be incorporated into P. If we simply add it, then the resulting program P' has the answer set

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

Thus, $query(index)$ is retracted and $query(cite_seer)$ is added. Simply adding a new piece of information to a logic program is, in general, not a suitable strategy for updating it, since conflicts may arise. We discuss some approaches to updates 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 of the knowledge base and used to compute 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 of the knowledge base, preference information can be useful to facilitate 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, if a rule with low priority is retracted, then the resulting program has an answer set).

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 be obtained 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, 6.8), usually by means of

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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, and also 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, and also 8.1); on the other hand, when priorities are expressed in the object language (sections 6.1, 6.4, 6.5, 6.7, 6.8), the preference relation can itself be dynamically revised or updated together with the remaining knowledge.

Fixed vs. variable strategies. 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 also 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 easily be modified for other purposes.

6.1 Prioritized defaults by Gelfond and Son

In Gelfond and Son (1998), 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 the form 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.

A simple example from Gelfond and Son (1998) shows that the formalism works as intended. Consider the following rules:

$$\begin{aligned} r_1: &\quad \neg flies(tweety) \leftarrow not\ flies(tweety), penguin(tweety);\\ r_2: &\quad flies(tweety) \leftarrow not\ \neg flies(tweety), bird(tweety);\\ r_3: &\quad penguin(tweety) \leftarrow not\ \neg penguin(tweety), bird(tweety); \end{aligned}$$

together with the fact $bird(tweety) \leftarrow$, and with the information that r_1 is preferred over r_2, r_2 over r_3, and r_1 over r_3. Then, the approach computes only one answer set, containing the literals $bird(tweety), penguin(tweety), \neg flies(tweety)$, as expected (cf. (Gelfond and Son, 1998) for the necessary technical definitions).

In this approach, defaults and rules are technically distinguished, as rules are non defeasible, while defaults express defeasible 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 require only simple changes in the formalism. Alternative strategies for priority handling can thus be implemented by necessary changes in the set of axioms. For more complex examples implementing these features, cf. Gelfond and Son (1998).

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

6.2 Preferred answer sets by Brewka and Eiter

In Brewka and Eiter (1999), 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 Brewka and Eiter (2000)).

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.

Consider again the Tweety example, expressed by the following program P, where the rules are listed with descending priority (cf. Brewka and Eiter, 1999):

$$\begin{aligned} P \;=\; \{ & penguin(tweety) \;\leftarrow\; ; \\ & bird(tweety) \;\leftarrow\; ; \\ & \neg flies(tweety) \;\leftarrow\; not\ flies(tweety), penguin(tweety); \\ & flies(tweety) \;\leftarrow\; not\ \neg flies(tweety), bird(tweety)\}. \end{aligned}$$

Then, P has a unique preferred answer set, containing, as intended, the literals $penguin(tweety), bird(tweety), \neg flies(tweety)$.

A *strong* and a *weak* notion of preferred answer sets is considered, respectively, where the second one ensures that if a program has an answer set, then it also has at least a preferred one, which is not guaranteed with the strong notion, 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 information 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 Inoue and Sakama (1996), the problem of reasoning with explicit representation of priorities in logic programming is addressed. The proposed 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* (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 answer set semantics.

In the framework of PLP, the usual default reasoning problem about Tweety can

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

be represented by the following program P (cf. Inoue and Sakama, 1996):

$$\begin{aligned} \neg flies(tweety) &\leftarrow penguin(tweety); \\ bird(tweety) &\leftarrow penguin(tweety); \\ penguin(tweety) &\leftarrow ; \\ flies(tweety) &\leftarrow default(tweety), bird(tweety); \\ default(tweety) \vee not\ default(tweety) &\leftarrow , \end{aligned}$$

together with the preference information $not\ default(tweety) \prec default(tweety)$, indicating that $not\ default(tweety)$ is preferred over $default(tweety)$. The unique preferred answer set is thus $\{bird(tweety), penguin(tweety), \neg flies(tweety)\}$, as desired.

Priorities over literals of 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; examples can be found in Inoue and Sakama (1996).

In Inoue and Sakama (1999), abduction is used to derive appropriate priorities yielding 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 in terms of an integrated framework for the abduction of both literals and priorities, using prioritized and abductive ELPs.

The method can be described as follows. 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, if the given observation is credulously but not skeptically inferred by the initial program. In a second step, an extended framework is obtained by combining preference abduction with ordinary abduction, providing the given observation cannot even be credulously 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 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, and if 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 original framework. Moreover, the extended 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 arrive at 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 Foo and Zhang (1997a), PLPs with static priorities under an extended answer set semantics are introduced. In this approach, a PLP consists of an ELP, a naming

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

function for rules, and a strict partial order over 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 the form of an ELP with no priority relation. The answer sets of this program are the intended answer sets of the initial PLP.

As a formalization of the usual example about Tweety, consider the following program P (cf. Foo and Zhang, 1997a):

$$\begin{aligned} \mathit{flies}(\mathit{tweety}) &\leftarrow \mathit{bird}(\mathit{tweety}), \mathit{not}\ \neg\mathit{flies}(\mathit{tweety});\\ \neg\mathit{flies}(\mathit{tweety}) &\leftarrow \mathit{penguin}(\mathit{tweety}), \mathit{not}\ \mathit{flies}(\mathit{tweety});\\ \mathit{bird}(\mathit{tweety}) &\leftarrow ;\\ \mathit{penguin}(\mathit{tweety}) &\leftarrow , \end{aligned}$$

where the second rule has higher priority than the first one. The single answer set of the program is given by

$$\{\neg\mathit{fly}(\mathit{tweety}), \mathit{bird}(\mathit{tweety}), \mathit{penguin}(\mathit{tweety})\}.$$

In Foo and Zhang (1997a), PLPs are extended to express dynamic priorities as well. The semantics of a dynamic PLP is defined in terms of the answer sets of a corresponding (static) PLP. More specifically, a dynamic PLP is a pair consisting of a program over a language containing a symbol expressing a partial order between names of rules, and a naming function for rules. To ensure the consistency of the given partial order, 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 Delgrande *et al.* (2000c), a methodology is introduced for computing the answer sets of ELPs with dynamic priorities between rules, explicitly expressed in the program itself (such a program is called *ordered*). On the basis of the encoded preferences, an ordered program is transformed into a standard ELP whose answer sets correspond to the intended preferred answer sets of the original one. The

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

transformation is realized by decomposing each rule of the program into a group of associated rules, in which newly introduced control atoms guarantee that successive rule applications are compatible with the given preference information.

The approach is sufficiently general to allow the specification of preferences among preferences, preferences holding in a particular context, and preferences holding by default. Moreover, the concept of *static preferences*, where the preference information about rules is fixed (e.g., given at the meta level like in most approaches to preference handling) is easily realizable.

Alternative strategies for preference handling can be achieved as well by changing the specification of the transformed program. For instance, in Delgrande *et al.* (2000a) it is shown that the method of Brewka and Eiter (1999) can be encoded within this framework. Furthermore, it is straightforward implementing the methodology, since the result of the transformation is a standard ELP and therefore existing logic programming systems can be used as underlying reasoning engines. The description of such an implementation is given in Delgrande *et al.* (2000b).

6.6 Disjunctive programs with inheritance by Buccafurri et al.

In Buccafurri *et al.* (1999a, 2001), a framework for disjunctive logic programs with inheritance (denoted $DLP^{<}$) on the basis of the 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 resolved by favoring rules which are more specific according to the hierarchy. The hierarchy itself 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 together with a partial order over them. The program for a given object in the hierarchy is then given by the collection of programs at and above the corresponding indexed program. The semantics of inheritance programs is defined as an extension of the standard answer set semantics.

6.7 Preference default theories by Brewka

In Brewka (2000), 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, viewed 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 definition of theory extensions. In a second step, a notion of prioritized inference is introduced, as the least fixed-point of an appropriate monotone operator, thus identifying epistemic states with preferential default theories under this ad hoc semantics. Due to a potential self-referentiability of preference information, not all theories expressed in this formalism possess extensions (i.e., acceptable sets of beliefs).

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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

- a single set is considered containing all information, in which there is no absolutely unrevisable information; and
- preferences 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 preferred extension of a preference default theory is given by 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.

For illustration, consider how the usual Tweety example works in this framework (cf. Brewka, 2000). Let T be the following theory (where $\supset$ stands for implication):

$$\begin{aligned} f_1 &: \ bird(tweety) \supset flies(tweety) \\ f_2 &: \ penguin(tweety) \supset \neg flies(tweety) \\ f_3 &: \ bird(tweety) \wedge penguin(tweety) \\ f_4 &: \ f_3 < f_1 \\ f_5 &: \ f_2 < f_1 \end{aligned}$$

where f_4 intuitively states that f_3 is preferred over f_1, and f_5 states that f_2 is preferred over f_1. Then, the only accepted conclusions of T are, as intended, given by the set $\{bird(tweety), penguin(tweety), \neg flies(tweety)\}$.

In section 7.8, we discuss the use of this formalism for representing revision strategies.

6.8 Ordered logic programs

Another approach for resolving conflicts between rules is given by ordered logic programs (Laenens and Vermeir, 1990; Gabbay *et al.*, 1991; Buccafurri *et al.*, 1996; Buccafurri *et al.*, 1999b) (cf. also section 7.9).

Ordered logic programs offer the possibility of specifying preferences between (sets of) rules at the object level, by means of a naming function for rules and of special atoms expressing priority between rule names. These special atoms can appear everywhere in a rule, thus permit us to express dynamic preferences.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Considering again the example about Tweety, it can be formalized by the program P containing the following rules:

$$\begin{aligned}
r_1 &: \quad penguin(tweety) \leftarrow \\
r_2 &: \quad bird(tweety) \leftarrow \\
r_3 &: \quad flies(tweety) \leftarrow bird(tweety), not\ \neg flies(tweety) \\
r_4 &: \quad \neg flies(tweety) \leftarrow penguin(tweety), not\ flies(tweety) \\
r_5 &: \quad r_3 \prec r_4 \leftarrow
\end{aligned}$$

where the last rule intuitively states that r_4 is preferred over r_3. Then, as intended, the preferred answer set of P is given by

$$\{penguin(tweety), bird(tweety) \neg flies(tweety)\}.$$

While ordered logic programs are similar to logic programs with inheritance as defined by Buccafurri *et al.* (2001), there is a salient difference: in the former, contradictions between rules r_1 and r_2 from different sets of rules R_1 and R_2, respectively, with no preference among them are removed, while in the latter they lead to inconsistency. For example, for $R_1 = \{rain \leftarrow\ \}$ and $R_2 = \{\neg rain \leftarrow\ \}$ having lower priority than $R_0 = \emptyset$, the respective ordered logic program has two preferred answer sets, $\{rain\}$ and $\{\neg rain\}$, while the respective inheritance program has no answer set.

6.9 Further approaches

Besides the methods discussed above, 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, as there is a one-to-one correspondence between the answer sets of an ELP P and the consistent extensions of a corresponding default theory (Gelfond and Lifschitz, 1991). A semantics for prioritized ELPs is thus inherited from any semantics for prioritized default logic, as several proposals discussed (Marek and Truszczyński, 1993; Brewka, 1994; Baader and Hollunder, 1995; Rintanen, 1998). 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 Brewka and Eiter (1999).

Further semantics for priorities in extended logic programming, which are not based on answer sets, have been proposed (Nute, 1994; Analyti and Pramanik, 1995; Brewka, 1996; Prakken and Sartor, 1997). They have quite different foundations such as logical entrenchment and specificity (Nute, 1994), reliability handling (Analyti and Pramanik, 1995), well-founded semantics (Brewka, 1996), or defeasible argumentation (Prakken and Sartor, 1997).

In Pradhan and Minker (1996), it is shown how priorities can be used to combine different potentially conflicting Datalog databases. Preferences are used to determine that information which is given up in the merging process. Three different semantics

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

of priorities are defined, two of which are equivalent. However, Pradhan and Minker do not consider negation (neither weak nor strong) at all, and the approach has thus limited expressive capability.

In Kowalski and Sadri (1991), 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 in which answer sets as defined in section 5.2 are also answer sets according to the definition of Kowalski and Sadri (1991). The main achievement is that programs which merely admit an inconsistent answer set (as possible in the definition of (Gelfond and Lifschitz, 1991)), become consistent in the new semantics. Thus, the approach in Kowalski and Sadri (1991) has more of a *contradiction removal method* than of a priority approach.

For further discussions and comparisons concerning different approaches for preferences, we refer the reader elsewhere (Brewka and Eiter, 1999; Delgrande and Schaub, 2000; Inoue and Sakama, 2000b).

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, e.g., be 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 (Alferes *et al.*, 1999a); 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 (Katsuno and Mendelzon, 1991). A detailed discussion on the different approaches to changes of the knowledge base can also be found elsewhere (Gärdenfors and Rott, 1995; Nebel, 1998); a clear distinction is not always possible, and 'hybrid' formalisms are possible as well (Eiter *et al.*, 2000a).

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 7.1–7.7, and 7.9), 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, viewed to represent changes in the external world, which the agent should be forced to accept (sections 7.1–7.8). Other formalisms assume no

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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, and 7.4–7.7). If the update policy is explicitly expressed, changes to the processing strategy (e.g., arbitrary priority assignments to the involved programs instead of fixed ones) could be easier realized. 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 the new knowledge base (sections 7.1–7.3, 7.5, and 7.6). 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, and 7.8), or rely on explicit priority information (section 7.6). 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 7.5–7.7).
- 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 information is 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 technically possible, i.e. the program resulting after an update can be directly subject to update again. (sections 7.1, 7.2, 7.4–7.8).

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

7.1 Revision programs by Marek and Truszczyński

In Marek and Truszczyński (1994), 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 should 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,

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 Alferes and Pereira (1997), 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 (Leite and Pereira, 1997; Alferes *et al.*, 1998; Alferes *et al.*, 2000a), 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. 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.

For illustration, we consider a simple example from Leite and Pereira (1997), which shows the difference of shifting inertia from literals to rules, as in Marek and Truszczyński (1994) and Alferes and Pereira (1997). Suppose the knowledge base is

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

given by the following program P:

$$\begin{aligned} go_home &\leftarrow not\ have_money; \\ go_restaurant &\leftarrow have_money; \\ have_money &\leftarrow , \end{aligned}$$

whose single model is $\{have_money, go_restaurant\}$. According to revision programming, updating this model with the information $out(have_money) \leftarrow in(robbed)$ and $in(robbed) \leftarrow$ results in the model $\{robbed, go_restaurant\}$, which is counterintuitive, as argued by Leite and Pereira (1997). On the other hand, updating P by applying inertia to rules, as realized in Leite and Pereira's approach, yields the desired model $\{robbed, go_home\}$.

In this sense, model updates are a special case of program updates, since models can be encoded as programs containing only facts. 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 the 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 case they are not overruled by the update program. This is achieved by defining a subprogram of the initial program containing those rules which should persist 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 then given, taking as input the initial and the update program, and producing an updated program whose models are the required updates, by introducing suitable inertia rules.

7.3 Abductive updates by Inoue and Sakama

Inoue and Sakama (1995) 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 (Moore, 1985), 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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 elaborated in Sakama (2000).

The framework of extended abduction is then used in Inoue and Sakama (1999) to model updates of nonmonotonic theories which are represented by ELPs. The following example from Alrefes *et al.* (2000a) shows what the results of the approach are. Consider the initial knowledge base given by the program P:

$$\begin{array}{rcll} P & = & \{watch_tv & \leftarrow \quad tv_on; \\ & & sleep & \leftarrow \quad not\ tv_on; \\ & & tv_on & \leftarrow \quad \} \end{array}$$

and the update expressed by the program U:

$$\begin{array}{rcll} U & = & \{power_failure & \leftarrow \quad ; \\ & & \neg tv_on & \leftarrow \quad power_failure\ \}. \end{array}$$

The update program resulting by the abductive representation of the problem has the unique (under minimality of change) answer set $\{power_failure, \neg tv_on, sleep\}$.

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 of 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 the 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 a 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 Inoue and Sakama (1995). 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 Foo and Zhang (1997b), 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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 Foo and Zhang (1998), the problem of updates is addressed if both old and new knowledge is encoded as ELP. 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 resolved. 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.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

7.5 Dynamic logic programming by Alferes et al.

Dynamic Logic Programming (DynLP) is introduced in Alferes *et al.* (1998) as a new paradigm, which extends the idea of an update of a logic program P by means of a logic program U (denoted $P \oplus U$) to compositional sequences $\bigoplus P = P_1 \oplus P_2 \oplus ... \oplus P_n$.

The approach is obtained by defining a new extended language and by 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.

As an example, consider again the programs P and U from section 7.3 (Alferes *et al.*, 2000a):

$$\begin{array}{lrcl} P = & \{watch_tv & \leftarrow & tv_on; \\ & sleep & \leftarrow & not\ tv_on; \\ & tv_on & \leftarrow & \} \\ \\ U = & \{power_failure & \leftarrow & ; \\ & \neg tv_on & \leftarrow & power_failure\ \}. \end{array}$$

The unique stable model of the update of P by U is, as intended,

$$\{power_failure, sleep\}.$$

Performing a second update by the rule $not\ \ power_failure \leftarrow$, the new stable model becomes then $\{tv_on, watch_tv\}$.

In Alferes *et al.* (2000a), 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 to prevent contradiction.

Yet another kind of contradiction is an implicit one: when the updated program is normal, explicit contradiction cannot arise, but it may well be that the updated

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 Dell'Acqua and Pereira (1999b) with agents based on the paradigm of Kowalski and Sadri (1996) (Dell'Acqua *et al*., 1998), 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 Alferes and Pereira (2000), an integrated framework combining updates and preferences is introduced, based on DynLP and the preference semantics due to Brewka and Eiter (1999). In this approach, a new language is defined, modeling sequences of programs $P_1, \ldots, P_n$ resulting from 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 updates eliminate rules: preferences eliminate less preferred rules, selecting among the available stable models, and updates eliminate rules overruled by other ones, 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) according to the method of Brewka and Eiter (1999) 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.

To facilitate the capturing of both preferences and updates in a single framework,

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 Brewka and Eiter (1999). This is achieved in the following way: while in Brewka and Eiter (1999), 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 Brewka and Eiter (1999).

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 Alferes and Pereira (2000), 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 with evolution by means of updates and evolution of the priority relation. Moreover, the language 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 $\langle P_i, Q_i\rangle$ consists of a program P_i representing rules and of a program Q_i describing a priority relation among rules. In general, as an update of the priority relation may depend on some other predicates, rules in programs describing priorities are allowed to refer to predicates defined in the programs representing knowledge.

Let us consider an example from Alferes and Pereira (2000). The initial knowledge is given by the program P_1 containing the facts $safe(chevrolet)$, $fast(chevrolet)$, $expensive(chevrolet)$, $safe(volvo)$, $fast(porsche)$, together with the following rules:

$$\begin{aligned}
r_1 &: \ not\ buy(X) \leftarrow avoid(X);\\
r_2 &: \ avoid(X) \leftarrow not\ buy(X), expensive(X);\\
r_3 &: \ buy(X) \leftarrow not\ fast(X);\\
r_4 &: \ avoid(Y) \leftarrow fast(X), buy(X), Y \neq X,
\end{aligned}$$

where the preferences over rules are as follows: $r_2 < r_3, r_2 < r_4$, stating that rule r_2 is preferred over both r_3 and r_4. Let us then update this knowledge by the program P_2, made up of the following rules:

$$\begin{aligned}
r_5 &: \ buy(X) \leftarrow not\ avoid(X), safe(X);\\
r_6 &: \ avoid(Y) \leftarrow safe(X), buy(X), Y \neq X,
\end{aligned}$$

with associated priorities $r_5 < r_3, r_5 < r_4, r_6 < r_3, r_6 < r_4, r_2 < r_5, r_2 < r_6$. Then, the only preferred stable model of the update with priorities is

$$\{buy(volvo), avoid(porsche), avoid(chevrolet)\}.$$

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Besides the extension of DynLP to consider preferences, the work in Dell'Acqua and Pereira (1999b) about updating agents is extended for preference and updating in multi-agent systems (Dell'Acqua and Pereira, 1999a). 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

Concerning the approach discussed in section 6.6, it is natural to view a sequence of updates as an inheritance program where later updates are considered to be more specific. More specifically, as argued in Eiter *et al.* (2000a), we may view the sequence $P = P_1, \ldots, P_n$ of programs as a knowledge base with inheritance order $P_n < P_{n-1} < \ldots < P_2 < P_1$, i.e., for each $j = 1, \ldots, i$, P_{j+1} is more specific than P_j. It is shown (Eiter *et al.*, 2000b) that for certain classes of programs, the update sequence $P_1, \ldots, P_n$ is equivalent to such a corresponding inheritance program.

For illustration, consider again the example from Alrefes *et al.* (2000a), but in a slightly adapted form. Let the initial knowledge base be given by the following program P:

$$\begin{array}{rcl} P = \{watch_tv & \leftarrow & tv_on; \\ sleep & \leftarrow & not\ tv_on; \\ night & \leftarrow & ; \\ tv_on & \leftarrow & \} \end{array}$$

and the update defined by the program U:

$$\begin{array}{rcl} U = \{power_failure & \leftarrow & ; \\ \neg tv_on & \leftarrow & power_failure\ \}. \end{array}$$

The update of P by U has, as intended, the unique answer set

$$S = \{power_failure, \neg tv_on, sleep, night\}.$$

If, again, new knowledge arrives in form of the program $U' = \{\neg power_failure \leftarrow\ \}$, then the unique answer set of the sequence P, U, U' is

$$S' = \{\neg power_failure, tv_on, watch_tv, night\}.$$

It is easy to see that in the formalism of Buccafurri *et al.* (n.d.), S is an answer set of the inheritance program $U < P$, and, similarly, S' is an answer set of the inheritance program $U' < U < P$.

As shown in Alferes *et al.* (2000a), dynamic logic programs naturally generalize the idea of updating interpretations, viewed as collections of positive and negative facts, through Marek and Truszczyński's revision programs (Marek and Truszczyński, 1994) (which, in turn, have been further considered by Przymusinski and Turner (1997)). Thus, inheritance programs also generalize interpretation updates.

Buccafurri *et al.* (2001) also contains a transformation from disjunctive inheri-

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

tance 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 Brewka (2000), a nonmonotonic framework for belief revision is introduced 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 the newly added information into account, possibly containing preference information as well.

The suggested notion of revision is evaluated with respect to the AGM postulates (Alchourrón *et al.*, 1985) (suitably reformulated in terms of belief sets). Furthermore, it is explained why it is meaningful that some postulates are not satisfied by the approach.

Contraction (i.e., making a formula underivable) can be handled in the case that the agent receives explicit 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 to be necessary to make the formalism be able to deal with it as well. 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 some notion of utility over information.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 (Buccafurri *et al.*, 1999a; Buccafurri *et al.*, 2001) and extensions of dynamic logic programming (Alferes *et al.*, 2000a; Alferes *et al.*, 2000b), 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 a satisfactory behavior, and, in particular, simple inconsistencies between knowledge bases of which each one is consistent (like $\{a \leftarrow\}$ and $\{\neg a \leftarrow\}$) may lead to inconsistency of the combined knowledge base. On the other hand, ordered logic programming (Laenens and Vermeir, 1990; Gabbay *et al.*, 1991; Buccafurri *et al.*, 1996; Buccafurri *et al.*, 1999b) can avoid this problem. The notion of a stable model (Buccafurri *et al.*, 1996; Buccafurri *et al.*, 1999b) 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 (Buccafurri *et al.*, 1999b) (cf. also section 6.8).

Observe that specific approaches for combining classical propositional and first-order knowledge bases have been defined (Baral *et al.*, 1991b; Revesz, 1993; Baral *et al.*, 1991a; Liberatore and Schaerf, 1998), while (Baral *et al.*, 1994) has addressed the problem of combining default logic knowledge bases which must satisfy some constraints. ELPs inherit this method by the correspondence between ELPs and default logic theories. Roughly speaking, in terms of ELPs, the approach tries to retain as much as possible from the union of the sets of rules in the given programs $P_1, \ldots, P_n$ such that consistency is retained and all given constraints are satisfied. At the heart of the approach lies a contradiction removal method for conflicts that arise between consistent default theories of equal priority. The authors also extend their method to prioritized merging of default theories.

Axioms for arbitration operators in the context of classical propositional knowledge bases have been addressed (Revesz, 1993; Liberatore and Schaerf, 1998). In Revesz (1993), 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 the AGM theory (Alchourrón *et al.*, 1985) and with update operators according to the theory of Katsuno and Mendelzon (1991). On the other hand, in Liberatore and Schaerf (1998), axioms for arbitration operators are considered such that, as argued, belief revision can be easily formulated through arbitration but, conversely, a 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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 (van Emden, 1986), 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 (Marek and Truszczyński, 1999), others with rule heads or rule bodies (Niemelä *et al*., 1999; Lukasiewicz, 1998; Lukasiewicz, 2001), or with single atoms of the language (Ng, 1997), or with formulas (Ng and Subrahmanian, 1992). The information is sometimes expressed by means of a single value (Marek and Truszczyński, 1999), sometimes in the form of ranges or intervals of values (Ng and Subrahmanian, 1992; Ng and Subrahmanian, 1993; Niemelä *et al*., 1999; Lukasiewicz, 1998; Lukasiewicz, 2001), 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 distributions 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 Buccafurri *et al*. (2000). 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 proposed 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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 Niemelä *et al.* (1999), 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 (Gelfond and Lifschitz, 1988).

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 Marek and Truszczyński (1999), 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 *reusability* and *no-reusability*, respectively. In the reusability approach, the cost of applying rules for deriving an atom is paid only once, and then the derived atom can be 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 minimal 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, Ng, Subrahmanian and Dekhtyar develop a theory of probabilistic logic programming (Ng and Subrahmanian, 1992, 1993, 1994; Dekhtyar and Subrahmanian, 2000) based on the work about annotated logic (Subrahmanian, 1987; Kifer and Subrahmanian, 1992), and on a possible-worlds semantics. Alternatively, probabilities can refer to the elements in the domain, and represent statistical

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

knowledge. We do not discuss this approach here; for more information about this topic, cf. Ng (1997), Poole (1993), Bacchus *et al.* (1993) and Pearl (1989).

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 (Ng and Subrahmanian, 1993), which is then extended to handle default reasoning under the stable model semantics (Ng and Subrahmanian, 1994). This approach to probabilistic logic programming is used in (Dix *et al.*, 2000b) to extend the framework for agent programming presented in Dix *et al.* (2000b), 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 Subrahmanian *et al.* (2000) 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-worlds semantics is discussed in Lukasiewicz (1998, 2001), 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. Of particular interest is reasoning about the temporal evolution of the agent's knowledge base and its behavior, depending on an underlying update policy or other principles. This poses specific requirements both on the knowledge representation language, which must allow the possibility to express temporal features, and on the reasoning mechanism.

The topic may be considered from different perspectives. Temporal aspects are relevant when considering agent actions and their effects, especially in conjunction with beliefs, desires and intentions held 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 (Alferes *et al.*, 1999a), 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 providing a pointer concerning how temporal logics and model checking techniques could be useful in conjunction with LUPS for reasoning over updates. A full coverage of all fields of research relevant to temporal reasoning and knowledge bases, however, is beyond the scope of our paper. In particular, the following important research areas are omitted in our review:

- dynamic logic (Harel, 1979);
- process logic, as described in Harel *et al.* (1982), among others;

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

- event calculus (Kowalski and Sergot, 1986);
- situation calculus, tracing back to McCarthy and Hayes (Turner, 1997; Pinto and Reiter, 1993; Pinto and Reiter, 1995), and its relationship with the event calculus (Kowalski and Sadri, 1997);
- transaction logic and transaction logic programming (Bonner and Kifer, 1994);
- action logic (Pratt, 1990).

A comprehensive analysis of these and other related topics can be found in Bonner and Kifer (1998).

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 concurrency (Baral and Gelfond, 1997) and ramifications (Giunchiglia *et al*., 1997) into account. In this field, much work has been devoted to describing and discussing the features of alternative action languages (Gelfond and Lifschitz, 1998; Lifschitz, 1999), together with the corresponding reasoning mechanisms, combining them with temporal action logics or with the situation calculus (Giunchiglia and Lifschitz, 1999). A review of the relevant tasks for reasoning agents in such dynamic domains can be found in Baral and Gelfond (2000).

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 Alferes *et al.* (1999b).

9.2 Temporal logics for BDI agents

Temporal aspects are addressed in BDI agents (*belief*, *desire*, *intention*; e.g., see Georgeff and Rao, 1991) by means of an extension of the branching-time temporal logic CTL* (Emerson and Srinivasan, 1988). 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 is to be performed. A set of belief-accessible worlds is associated with each state, 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 actual 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.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

For an extensive treatment of this subject, we refer the reader to Wooldridge's excellent monograph (Wooldridge, 2000).

9.3 LUPS, a language for specifying updates

In Alferes *et al.* (1999a), the language LUPS for specifying update sequences of logic programs has been introduced. 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 making each update program in a given sequence of programs depend on some specifications and conditions. Furthermore, 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 class of all epistemic states of 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 a 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 discussed in section 3, in the following we 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.

To keep track about the different approaches evaluated below, Tables 1 and 2 summarize the main features of each analyzed formalism.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Table 1. *Approaches for preference handling*

Approach	Section	Reference	Features
Prioritized Defaults	6.1	(Gelfond and Son, 1998)	Dynamic priorities over rules at the object level, plus axiom rules
Preferred Answer Sets	6.2	(Brewka and Eiter, 1999)	Static order over rules, with ad hoc semantics
Prioritized Programs	6.3	(Inoue and Sakama, 1996) (Inoue and Sakama, 1999)	Static priority relation over literals, with ad hoc semantics
Prioritized Programs	6.4	(Foo and Zhang, 1997a)	Static or dynamic order over rules, with ad hoc semantics
Compiled Preferences	6.5	(Delgrande *et al.*, 2000c)	Dynamic priorities over rules at the object level, with program rewriting
Inheritance Programs	6.6	(Buccafurri *et al.*, 2001) (Buccafurri *et al.*, 1999a)	Static order over rules, with program rewriting
Preference Theories	6.7	(Brewka, 2000)	Dynamic priorities over rules at the object level, with ad hoc semantics
Ordered LPs	6.8	(Laenens and Vermeir, 1990) (Gabbay *et al.*, 1991) (Buccafurri *et al.*, 1999b)	Dynamic priorities over rules at the object level

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, 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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 to 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. For example, 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, 6.7 and 6.8.

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. Observe, however, that preferences must be used with care in the respective approaches. While the approach in section 6.8 implicitly supports removal of conflicts between contradicting information from different information sources (viewed as sets of rules) of equal priority, such a mechanism must in the other approaches be achieved through proper representation of the knowledge, to avoid inconsistency.

Approaches for handling preferences that are based on standard semantics rather than on *ad hoc* semantics offer some advantages:

1. they are easier to be compared with other approaches;
2. their behavior can be better understood;
3. they are easier to be integrated or extended to other frameworks.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 preferably be encoded explicitly in the object language of the knowledge base, in the 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. For instance, the way in which priorities are handled for determining a user's profile may be different from the way in which they are handled for choosing 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 the form of a single rule or a set of rules) comes from another agent and should be incorporated in the current 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 more easily 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, for instance, 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 Eiter *et al.* (2000a) 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 Inoue and Sakama (1999), 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 Brewka and Eiter, 1999) and particular properties of the approaches as to the behavior of the corresponding agent should

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

be better analyzed. Interesting is also the problem of self-reference of the preference relation, as pointed out in Brewka (2000), 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 a relative order among rules. For example, this is the case in weighted logic programming (Marek and Truszczyński, 1999) and also, to some extent, in disjunctive logic programming with weak constraints (Buccafurri *et al.*, 2000), 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 the form of a range identified by lower and upper bounds (Niemelä *et al.*, 1999).

Atom level: the quantitative information can be associated with atoms in the 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 Dekhtyar and Subrahmanian (2000), 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, for example, 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 Dix *et al.* (2000b), 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 Lukasiewicz (1998). 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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Table 2. *Approaches for revision and update*

Approach	Section	Reference	Features
Revision Programs	7.1	(Marek and Truszczyński, 1994)	Revision of a set of facts by a set of rules
Update Programs	7.2	(Alferes and Pereira, 1997) (Leite and Pereira, 1997)	Revision of a LP by a LP, through rewriting
Abductive Updates	7.3	(Inoue and Sakama, 1995) (Inoue and Sakama, 1999)	Theory update by a set of facts
Updates by Priorities	7.4	(Foo and Zhang, 1997b) (Foo and Zhang, 1998)	Update of a set of facts or a LP by a LP, through rewriting
Dynamic Programs	7.5	(Alferes *et al*., 1998) (Alferes *et al*., 2000a)	Update by a (sequence of) LPs, through rewriting
Updates plus Preferences	7.6	(Alferes and Pereira, 2000)	Update by a (sequence of) LPs plus preferences
Updates by Inheritance	7.7	(Buccafurri *et al*., 2001)	Update by a (sequence of) LPs, through rewriting
Revision of Preference Theories	7.8	(Brewka, 2000)	Revision by a single rule/fact, ad hoc semantics
Arbitration	7.9	Different proposals	Merging of two knowledge bases

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

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 the form of a single rule or fact, or in the 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 communicated 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, usually assigning 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 level.

It would also be useful to have the possibility of explicitly specifying which of the knowledge bases to be merged has higher priority, since there may be application domains in which old information is to be preserved against new one. Moreover, 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 Eiter *et al.* (2000a)). The rewriting technique makes the update policy explicit in the resulting knowledge base, and can be usually modified or extended to encode other strategies or update mechanisms. This could be interesting in order to differentiate

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

the update process on the basis of the source of the update information, which could be, e.g. another information agent, or one of the data sources communicating changes in its content, or an internal learning procedure. 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 such a 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 proposed framework of Alferes and Pereira (2000), 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 among the rules who 'survived' the update. A more general approach would be useful.

Another method which could be used for a uniform handling of updates (as explained in section 7.7) and preferences is disjunctive logic programming with inheritance (Buccafurri *et al*., 2001), although it is originally conceived as a mechanism for representing inheritance hierarchies, thus not being appropriate for every sort of preference relation (cf. the considerations in section 10.1).

Other approaches addressing both updates and preferences have hitherto not been proposed. A possibility would be to extend an approach such as the one in section 6.5, 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 (Eiter *et al*., 2000a). 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, 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 (like the formalism of section 7.5), and additional mechanisms or changes to the update process are required to at least ensure 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 rele-

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

vance. 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 proposed, which express minimality requirements and are encoded in the update process (Eiter *et al.*, 2000a). 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 Eiter *et al.* (2000a)). 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 Brewka (2000), is *forgetting* of rules. Since the knowledge base of an actual 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, like 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 proposed for *vacuuming* relational databases, which is a non-trivial task.

Other improvements or extensions of the formalisms are clearly significant for information agents. An interesting feature is the one suggested in Alferes *et al.* (2000b), namely the capability of performing updates along several dimensions (e.g., different time periods, different hierarchies) at the same time. This would especially be interesting for combining 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 Eiter *et al.* (2000a) is a first attempt in this direction.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 (Lifschitz, 1999; Cimatti and Roveri, 1999). 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 (Giunchiglia *et al.*, 1997)) effects;
- the possible results of concurrent actions (Baral and Gelfond, 1997);
- uncertainties about the effective state of the world (Georgeff and Rao, 1991);
- events in the world not known to the agent in advance and possibly affecting its decisions.

A number of *action languages* have been developed (Gelfond and Lifschitz, 1998; Lifschitz, 1999), and their expressivity and the possibility of querying knowledge at different stages of the evolution has been discussed.

On the other hand, LUPS (Alferes *et al.*, 1999a), 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 approaches along the first point of view of temporal reasoning often employ the situation calculus, an interesting tool for the second approach could be found in temporal reasoning and in model checking (Cimatti *et al.*, 1998; Cimatti and Roveri, 1999), choosing a branching-time logic similar to the BDI framework (Georgeff and Rao, 1991). 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.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

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 concerning 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, revision and update, expressing quantitative information, and 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 now be 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 proposed 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 both qualitative and quantitative decision making process. A proposal, though not strictly concerned with logic programming, is outlined in Dix *et al.* (2000b).

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 (Hunter, 2000a; Hunter, 2000b), description logics (Calvanese *et al.*, 1998; Calvanese *et al.*,

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

1999), object oriented formalisms (Abiteboul *et al.*, 1994), and automated theorem proving and machine learning techniques (Thomas, 1998, 1999).

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 on 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 for 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 (Foo and Zhang, 1998; Alferes and Pereira, 2000; Alferes *et al.*, 2000b), partly based on the approach suggested in Brewka and Eiter (1999), 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 the form of LUPS programs (Alferes *et al.*, 1999a) 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 (Emerson, 1996), and could take advantage of techniques and tools developed in the field of model checking (Clarke *et al.*, 2000) or planning (Cimatti *et al.*, 1998; Cimatti and Roveri, 1999). 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 specification of its evolution, is to know whether the agent behaves correctly, and 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.

Acknowledgments

We would like to thank the anonymous referees for their constructive comments which helped to improve this paper. This work was partially supported by the

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Austrian Science Fund (FWF) under grants P13871-INF, P14781-INF, and Z29-INF.

References

Abiteboul, S., Christophides, V., Cluet, S. and Scholl, M. (1994) From structured documents to novel query facilities. In: Snodgrass, R. and Winslett, M. (eds.), *Proc. 1994 ACM SIGMOD Int. Conf. on Management of Data*, pp. 313–324. ACM Press.

Abiteboul, S., McHugh, J., Quass, D., Widom, J. and Goldman, R. (1997a) Lore: A database management system for semistructured data. *SIGMOD Record*, **26**(3), 54–66.

Abiteboul, S., McHugh, J., Quass, D., Widom, J. and Wiener, J. (1997b) The Lorel query language for semistructured data. *International Journal on Digital Libraries*, **1**(1), 68–88.

Adali, S. and Emery, R. (1995) A uniform framework for integrating knowledge in heterogeneous knowledge systems. In: Chen, L. and Yu, P. (eds.), *Proc. 11th Int. Conf. on Data Engineering* (*ICDE'95*), pp. 513–521. IEEE Computer Society.

Adali, S. and Subrahmanian, V. S. (1996) Amalgamating knowledge bases, III: Algorithms, data structures, and query processing. *Journal of Logic Programming*, **28**(1), 45–88.

Adali, S., Brink, A., Emery, R., Lu, J., Rajput, A., Rogers, T., Ross, R., Subrahmanian, V. S. and Ward, C. (1995) *HERMES: A Heterogeneous Reasoning and Mediator System*. Manuscript, URL: `http://www.cs.umd.edu/projects/hermes/publications/abstracts/hermes.html`.

Alchourrón, C., Gärdenfors, P. and Makinson, D. (1985) On the logic of theory change: partial meet functions for contraction and revision. *Journal of Symbolic Logic*, **50**, 510–530.

Alferes, J. and Pereira, L. (1997) Update-programs can update programs. In: Dix, J., Pereira, L. and Przymusinski, T. (eds.), *Proc. of Non-monotonic Extensions of Logic Programming* (*NMELP'96*)*: Lecture Notes in Computer Science 1216*, pp. 110–131. Springer-Verlag.

Alferes, J. and Pereira, L. (2000) Updates plus preferences. In: Aciego, M., de Guzmán, I., Brewka, G. and Pereira, L. (eds.), *Proc. 7th European Workshop on Logics in Artificial Intelligence* (*JELIA 2000*)*: Lecture Notes in Computer Science 1919*, pp. 345–360. Springer-Verlag.

Alferes, J., Leite, J., Pereira, L., Przymusinska, H. and Przymusinski, T. (1998) Dynamic logic programming. In: Cohn, A., Schubert, L. and Shapiro, S. (eds.), *Proc. 6th Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR'98*), pp. 98–111. Morgan Kaufmann.

Alferes, J., Pereira, L., Przymusinska, H. and Przymusinski, T. (1999a) LUPS – A language for updating logic programs. In: Gelfond, M., Leone, N. and Pfeifer, G. (eds.), *Proc. 5th Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*)*: Lecture Notes in Computer Science 1730*, pp. 162–176. Springer-Verlag.

Alferes, J., Pereira, L., Przymusinska, H., Przymusinski, T. and Quaresma, P. (1999b) Preliminary exploration on actions as updates. In: Meo, M. and Vilares-Ferro, M. (eds.), *Proc. 1999 Joint Conf. on Declarative Programming* (*AGP'99*), pp. 259–271. L'Aquila, Italy.

Alferes, J., Leite, J., Pereira, L., Przymusinska, H. and Przymusinski, T. (2000a) Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming*, **45**(1–3), 43–70.

Alferes, J., Leite, J. and Pereira, L. (2000b) Multi-dimensional dynamic logic programming. In: Sadri, F. and Satoh, K. (eds.), *Proc. CL-2000 Workshop on Computational Logic in Multi-agent Systems* (*CLIMA 2000*), pp. 17–26.

Analyti, A. and Pramanik, S. (1995) Reliable semantics for extended logic programs with rule prioritization. *Journal of Logic and Computation*, **5**(3), 303–324.

Arens, Y. (1990) Services and information management for decision support. *Proc. Annual AI Systems in Government Conf.* (*AISIG'90*). Washington, DC.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Arens, Y. and Knoblock, C. (1992) Planning and reformulating queries for semantically-modeled multidatabase systems. *Proc. 1st Int. Conf. on Information and Knowledge Managements* (*CIKM 1992*), pp. 92–101. Baltimore, MD.

Arens, Y., Chee, C., Hsu, C. and Knoblock, C. (1993) Retrieving and integrating data from multiple information sources. *International Journal of Cooperative Information Systems*, **2**(2), 127–158.

Arens, Y., Knoblock, C. and Shen, W. (1996) Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, **6**(2–3), 99–130.

Arisha, K., Eiter, T., Kraus, S., Ozcan, F., Ross, R. and Subrahmanian, V. S. (1999) IMPACT: A platform for collaborating agents. *IEEE Intelligent Systems*, **14**(2), 64–72.

Baader, F. and Hollunder, B. (1995) Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. *Journal of Automated Reasoning*, **15**(1), 41–68.

Bacchus, F., Grove, A. J., Halpern, J. Y. and Koller, D. (1993) Statistical foundations for default reasoning. In: Bajcsy, R. (ed.), *Proc. 13th Int. Joint Conf. on Artificial Intelligence* (*IJCAI'93*), pp. 563–569. Morgan Kaufmann.

Baral, C. and Gelfond, M. (1997) Reasoning about effects of concurrent actions. *Journal of Logic Programming*, **31**(1–3), 85–117.

Baral, C. and Gelfond, M. (2000) Reasoning agents in dynamic domains. In: Minker, J. (ed.), *Logic-based Artificial Intelligence*, pp. 257–279. Kluwer Academic.

Baral, C., Kraus, S., Minker, J. and Subrahmanian, V. S. (1991a) Combining knowledge bases consisting of first order theories. In: Ras, Z. and Zemankova, M. (eds.), *Proc. 6th Int. Symp. on Methodologies for Intelligent Systems* (*ISMIS'91*): *Lecture Notes in Computer Science 542*, pp. 92–101. Springer-Verlag.

Baral, C., Kraus, S. and Minker, J. (1991b) Combining multiple knowledge bases. *IEEE Trans. on Knowledge and Data Engineering*, **3**(2), 208–220.

Baral, C., Kraus, S. and Minker, J. (1994) Combining default logic databases. *International Journal of Intelligent and Cooperative Information Systems*, **3**(3), 319–348.

Bargmeyer, B., Fowler, J., Nodine, M. and Perry, B. (1999) Agent-based semantic interoperability in InfoSleuth. *SIGMOD Record*, **28**(1), 60–67.

Bayardo, R., Bohrer, B., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A. and Woelk, D. (1997) InfoSleuth: Semantic integration of information in open and dynamic environments (Experience Paper). In: Peckham, J. (ed.), *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 195–206. ACM Press.

Bohrer, W., Ngu, A. and Nodine, M. (1999) Semantic brokering over dynamic heterogeneous data sources in InfoSleuth. *Proc. 15th Int. Conf. on Data Engineering* (*ICDE'99*), pp. 358–365. IEEE Press.

Bondarenko, A., Dung, P.M., Kowalski, R. and Toni, F. (1997) An abstract argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, **93**, 63–101.

Bonner, A. and Kifer, M. (1994) An overview of transaction logic. *Theoretical Computer Science*, **133**(2), 205–265.

Bonner, A. and Kifer, M. (1998) The state of change: a survey. In: Freitag, B., Decker, H., Kifer, M. and Voronkov, A. (eds.), *Transactions and Change in Logic Databases, International Seminar on Logic Databases and the Meaning of Change: Lecture Notes in Computer Science 1472*, pp. 1–36. Springer-Verlag.

Bressan, S. and Goh, C. (1998) Answering queries in context. In: Andreasen, T., Christiansen, H. and Legind Larsen, H. (eds.), *Proc. 3rd Int. Conf. on Flexible Query Answering Systems* (*FQAS'98*): *Lecture Notes in Computer Science 1495*, pp. 68–82. Springer-Verlag.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Bressan, S., Goh, C., Fynn, K., Jakobisiak, M., Hussein, K., Kon, H., Lee, T., Madnick, S., Pena, T., Qu, J., Shum, A. and Siegel, M. (1997) The COntext INterchange mediator prototype. In: Peckham, J. (ed.), *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 525–527. ACM Press.

Brewka, G. (1994) Adding priorities and specificity to default logic. In: MacNish, C., Pearce, D. and Pereira, L. (eds.), *Proc. Europ. Workshop on Logics in Artificial Intelligence* (*JELIA'94*): *Lecture Notes in Computer Science 838*, pp. 247–260. Springer-Verlag.

Brewka, G. (1996) Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, **4**, 19–36.

Brewka, G. (2000) Declarative representation of revision strategies. *Proc. 14th European Conf. on Artificial Intelligence* (*ECAI 2000*), pp. 18–22.

Brewka, G. and Eiter, T. (1999) Preferred answer sets for extended logic programs. *Artificial Intelligence*, **109**(1–2), 297–356.

Brewka, G. and Eiter, T. (2000) Prioritizing default logic. In: Hölldobler, S. (ed.), *Intellectics and Computational Logic – Papers in Honor of Wolfgang Bibel*, pp. 27–45. Kluwer Academic.

Buccafurri, F., Faber, W. and Leone, N. (2001) Disjunctive logic programs with inheritance. *Theory and Practice of Logic Programming*. (To appear.) (Also available as: Technical report DBAI-TR-99-30, Institute of Information Systems, TU Vienna, 1999.)

Buccafurri, F., Leone, N. and Rullo, P. (1996) Stable models and their computation for logic programming with inheritance and true negation. *Journal of Logic Programming*, **27**(1), 5–43.

Buccafurri, F., Faber, W. and Leone, N. (1999a) Disjunctive logic programs with inheritance. In: De Schreye, D. (ed.), *Proc. 16th Int. Conf. on Logic Programming* (*ICLP'99*), pp. 79–93. MIT Press.

Buccafurri, F., Leone, N. and Rullo, P. (1999b) Semantics and expressiveness of disjunctive ordered logic. *Ann. of Math. and Artificial Intelligence*, **25**(3–4), 311–337.

Buccafurri, F., Leone, N. and Rullo, P. (2000) Enhancing disjunctive datalog by constraints. *IEEE Trans. Knowledge and Data Engineering*, **12**(5), 845–860.

Calvanese, D., De Giacomo, G. and Lenzerini, M. (1998) What can knowledge representation do for semi-structured data? *Proc. 15th Nat. Conf. on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conf.* (*AAAI/IAAI 1998*), pp. 205–210. AAAI Press/MIT Press.

Calvanese, D., De Giacomo, G. and Lenzerini, M. (1999) Representing and reasoning on XML documents: a description logic approach. *Journal of Logic and Computation*, **9**(3), 295–318.

Cannata, P., Huhns, M., Jacobs, N., Ksiezyk, T., Ong, K., Sheth, A., Singh, M., Tomlinson, C. and Woelk, D. (1997) The Carnot heterogeneous database project: implemented applications. *Distributed and Parallel Databases*, **5**(2), 207–225.

Cimatti, A. and Roveri, M. (1999) Conformant planning via model checking. In: Biundo, S. and Fox, M. (eds.), *Recent Advances in AI Planning, Proc. 5th Europ. Conf. on Planning* (*ECP'99*): *Lecture Notes in Computer Science 1809*, pp. 21–34. Springer-Verlag.

Cimatti, A., Roveri, M. and Traverso, P. (1998) Strong planning in non-deterministic domains via model checking. In: Simmons, R., Veloso, M. and Smith, S. (eds.), *Proc. 4th Int. Conf. on Artificial Intelligence Planning Systems* (*AIPS'98*), pp. 36–43. AAAI Press.

Clarke, E.M., Grumberg, O. and Peled, D. (2000) *Model Checking*. MIT Press.

Cohen, W. (1998a) Integration of heterogeneous databases without common domains using queries based on textual similarity. In: Haas, L. and Tiwary, A. (eds.), *Proc. ACM SIGMOD Int. Conf. on Management of Data* (*SIGMOD 1998*), pp. 201–212. ACM Press.

Cohen, W. (1998b) A Web-based information system that reasons with structured collections

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

of text. In: Sycara, K. and Wooldridge, M. (eds.), *Proc. 2nd Int. Conf. on Autonomous Agents*, pp. 400–407. ACM Press.

Decker, K., Sycara, K. and Williamson, M. (1997) Middle-agents for the Internet. *Proc. 15th Int. Joint Conf. on Artificial Intelligence* (*IJCAI'97*), **1**, 578–583. Morgan Kaufmann.

Dekhtyar, A. and Subrahmanian, V. S. (2000) Hybrid probabilistic programs. *Journal of Logic Programming*, **43**(3), 187–250.

Delgrande, J. and Schaub, T. (2000) Expressing preferences in default logic. *Artificial Intelligence*, **123**(1–2), 41–87.

Delgrande, J., Schaub, T. and Tompits, H. (2000a) A compilation of Brewka and Eiter's approach to prioritization. In: Aciego, M., de Guzmàn, I., Brewka, G. and Pereira, L. (eds.), *Proc. 7th Europ. Workshop on Logics in Artificial Intelligence* (*JELIA 2000*)*: Lecture Notes in Computer Science 1919*, pp. 376–390. Springer-Verlag.

Delgrande, J., Schaub, T. and Tompits, H. (2000b) A compiler for ordered logic programs. In: Baral, C. and Truszczynski, M. (eds.), *Proc. 8th Int. Workshop on Non-monotonic Reasoning* (*NMR 2000*).

Delgrande, J., Schaub, T. and Tompits, H. (2000c) Logic programs with compiled preference. In: Horn, W. (ed.), *Proc. 14th Europ. Conf. on Artificial Intelligence* (*ECAI 2000*), pp. 392–398. IOS Press.

Dell'Acqua, P. and Pereira, L. (1999a) *Preferring and Updating with Multi-Agents*. Draft.

Dell'Acqua, P. and Pereira, L. (1999b) Updating agents. In: Rochefort, S., Sadri, F. and Toni, F. (eds.), *Proc. ICLP'99 Workshop on Multi-agent Systems in Logic Programming* (*MAS'99*).

Dell'Acqua, P., Sadri, F. and Toni, F. (1998) Combining introspection and communication with rationality and reactivity in agents. In: Dix, J., del Cerro, L. and Furbach, U. (eds.), *Proc. European Workshop on Logics in Artificial Intelligence* (*JELIA'98*)*: Lecture Notes in Computer Science 1489*, pp. 17–32. Springer-Verlag.

Dimopoulos, Y. and Kakas, A. (2000) Information integration and computational logic. *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-net*, 1–41. (URL: `http://www.compulog.org/net/Forum/Supportdocs.html`.)

Dix, J. (1996) Semantics of logic programs: their intuitions and formal properties. An overview. In: Fuhrmann, A. and Rott, H. (eds.), *Logic, Action and Information – Essays on Logic in Philosophy and Artificial Intelligence*, pp. 241–327. DeGruyter.

Dix, J., Pick, G. and Subrahmanian, V. S. (2000a) Meta agent programs. *Journal of Logic Programming*, **46**(1–2), 1–60.

Dix, J., Nanni, M. and Subrahmanian, V. S. (2000b) Probabilistic agent programs. *ACM Trans. Computational Logic*, **1**(2), 208–246.

Dung, P. M. (1993) On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming. In: Bajcsy, R. (ed.), *Proc. 13th Int. Joint Conf. on Artificial Intelligence* (*IJCAI'93*), pp. 852–859. Morgan Kaufmann.

Duschka, O. and Genesereth, M. (1997) Query planning in Infomaster. *Proc. 12th Annual ACM Symposium on Applied Computing* (*SAC'97*), pp. 109–111.

Eiter, T. Subrahmanian, V. S. (1999) Heterogeneous active agents, II: Algorithms and complexity. *Artificial Intelligence*, **108**(1–2), 257–307.

Eiter, T., Pick, G. and Subrahmanian, V. S. (1999) Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, **108**(1–2), 179–255.

Eiter, T., Fink, M., Sabbatini, G. and Tompits, H. (2000a) Considerations on updates of logic programs. In: Aciego, M., de Guzmán, I., Brewka, G. and Pereira, L. (eds.), *Proc. 7th Europ. Workshop on Logic in Artificial Intelligence* (*JELIA 2000*)*: Lecture Notes in Artificial Intelligence 1919*, pp. 2–20. Springer-Verlag.

Eiter, T., Fink, M., Sabbatini, G. and Tompits, H. (2000b) *On Updates of Logic Programs:*

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Semantics and Properties. Technical Report, INFSYS RR-1843-00-08. Technische Universität Wien, Inst. für Informationssysteme, Abt. Wissensbasierte Systeme.

Emerson, E. (1996) Automated temporal reasoning about reactive systems. In: Moller, F. and Birtwistle, G. (eds.), *Logics for Concurrency: Structure versus Automata: Lecture Notes in Computer Science 1043*, pp. 41–101. Springer-Verlag.

Emerson, E. and Srinivasan, J. (1988) Branching time temporal logic. In: de Bakker, J., de Roever, W. and Rozenberg, G. (eds.), *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency* (*REX Workshop 1988*)*: Lecture Notes in Computer Science 354*, pp. 123–172. Springer-Verlag.

Finin, T., Fritzson, R., McKay, D. and McEntire, R. (1994) KQML as an agent communication language. *Proc. 3rd Int. Conf. on Information and Knowledge Management* (*CIKM'94*), pp. 456–463. ACM.

Flach, P. and Kakas, A. (2000) *Abduction and Induction – Essays on Their Relation and Integration.* Kluwer Academic.

Flores-Mendez, R. (1999) Towards a standardization of multi-agent system frameworks. *ACM Crossroads*, **5**(4), 18–24.

Foo, N. and Zhang, Y. (1997a) Answer sets for prioritized logic programs. In: Maluszynski, J. (ed.), *Proc. Int. Logic Programming Symp.* (*ILPS'97*), pp. 69–83. MIT Press.

Foo, N. and Zhang, Y. (1997b) Towards generalized rule-based updates. *Proc. 15th Int. Joint Conf. on Artificial Intelligence* (*IJCAI'97*), **1**, 82–88. Morgan Kaufmann.

Foo, N. and Zhang, Y. (1998) Updating logic programs. In: Prade, H. (ed.), *Proc. 13th Europ. Conf. on Artificial Intelligence* (*ECAI'98*), pp. 403–407. Wiley.

Fröhlich, P., Nejdl, W., Schroeder, M., Damásio, C. and Pereira, L. (1999) Using extended logic programming for alarm-correlation in cellular phone networks. In: Imam, I., Kodratoff, Y., El-Dessouki, A. and Ali, M. (eds.), *Multiple Approaches to Intelligent Systems, Proc. 12th Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* (*IEA/AIE'99*)*: Lecture Notes in Computer Science 1611*, pp. 343–352. Springer-Verlag.

Gabbay, D., Laenens, E. and Vermeir, D. (1991) Credulous vs. sceptical semantics for ordered logic programs. In: Allen, J., Fikes, R. and Sandewall, E. (eds.), *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR'91*), pp. 208–217. Morgan Kaufmann.

Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. and Widom, J. (1995) Integrating and accessing heterogeneous information sources in TSIMMIS. *Proc. AAAI Symp. on Information Gathering*, pp. 61–64.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V. and Widom, J. (1997) The TSIMMIS approach to mediation: data models and languages. *Journal of Intelligent Information Systems*, **8**(2), 117–132.

Gärdenfors, P. and Rott, H. (1995) Belief revision. In: Gabbay, D., Hogger, C. and Robinson, J. (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume IV: Epistemic and Temporal Reasoning*, pp. 35–132. Oxford Science Publications.

Gelfond, M. and Lifschitz, V. (1988) The stable model semantics for logic programming. In: Kowalski, R. and Bowen, K. (eds.), *Logic Programming, Proc. 5th Int. Conf. and Symp.* (*ICLP/SLP'88*), pp. 1070–1080. MIT Press.

Gelfond, M. and Lifschitz, V. (1991) Classical negation in logic programs and disjunctive databases. *New Generation Computing*, **9**(3–4), 365–386.

Gelfond, M. and Lifschitz, V. (1998) Action languages. *Electronic Trans. Artificial Intelligence*, **2**(3–4), 193–210.

Gelfond, M. and Son, T. (1998) Reasoning with prioritized defaults. In: Dix, J., Pereira, L. and Przymusinski, T. (eds.), *Logic Programming and Knowledge Representation, 3rd Int.*

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Workshop, Selected Papers (*LPKR'97*): *Lecture Notes in Computer Science 1471*, pp. 164–223. Springer-Verlag.

Genesereth, M. (1991) Knowledge Interchange Format. In: Allen, J., Fikes, R. and Sandewall, E. (eds.), *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR'91*), pp. 599–600. Morgan Kaufmann.

Genesereth, M., Keller, A. and Duschka, O. (1997) Infomaster: An information integration system. In: Peckham, J. (ed.), *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 539–542. ACM Press.

Georgeff, M. and Rao, A. (1991) Modeling rational agents within a BDI-architecture. In: Allen, J., Fikes, R. and Sandewall, E. (eds.), *Proc. 2nd Int. Conf. on Knowledge Representation and Reasoning* (*KR'91*), pp. 473–484. Morgan Kaufmann.

Giunchiglia, E. and Lifschitz, V. (1999) Action languages, temporal action logics and the situation calculus. *Linköping Electronic Articles in Computer and Information Science*, **4**, 1–7.

Giunchiglia, E., Kartha, G. and Lifschitz, V. (1997) Representing action: indeterminacy and ramifications. *Artificial Intelligence*, **95**(2), 409–438.

Harel, D. (1979) *First-order Dynamic Logic: Lecture Notes in Computer Science 68*. Springer-Verlag.

Harel, D., Kozen, D. and Parikh, R. (1982) Process logic: expressiveness, decidability, completeness. *J. of Computer and System Sciences*, **25**(2), 144–170.

Huhns, M. and Singh, M. (1992) The semantic integration of information models. *AAAI Workshop on Cooperation among Heterogeneous Intelligent Agents*.

Huhns, M. and Singh, M. (1994) Automating workflows for service provisioning: integrating AI and database technologies. *IEEE Expert*, **9**(5), 19–23.

Hunter, A. (2000a) Merging potentially inconsistent items of structured text. *IEEE Trans. Knowledge and Data Engineering*, **34**(3), 305–332.

Hunter, A. (2000b) Reasoning with inconsistency in structured text. *Knowledge Engineering Review*, **15**(2), 1–21.

Inoue, K. and Sakama, C. (1994) On the equivalence between disjunctive and abductive logic programs. In: Van Hentenryck, P. (ed.), *Proc. 11th Int. Conf. on Logic Programming* (*ICLP'94*), pp. 489–503. MIT Press.

Inoue, K. and Sakama, C. (1995) Abductive framework for nonmonotonic theory change. *Proc. 14th Int. Joint Conf. on Artificial Intelligence* (*IJCAI'95*), pp. 204–210. Morgan Kaufmann.

Inoue, K. and Sakama, C. (1996) Representing priorities in logic programs. In: Maher, M. (ed.), *Proc. Joint Int. Conf. and Symp. on Logic Programming* (*JICSLP'96*), pp. 82–96. MIT Press.

Inoue, K. and Sakama, C. (1999) Updating extended logic programs through abduction. In: Gelfond, M., Leone, N. and Pfeifer, G. (eds.), *Proc. 5th Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*): *Notes in Artificial Intelligence 1730*, pp. 147–161. Springer-Verlag.

Inoue, K and Sakama, C. (2000a) Abductive logic programming and disjunctive logic programming: their relationship and transferability. *Journal of Logic Programming*, **44**(1–3), 75–100.

Inoue, K. and Sakama, C. (2000b) Prioritized logic programming and its applications to commonsense reasoning. *Artificial Intelligence*, **123**(1–2), 185–222.

Jennings, N., Noriega, P., Parsons, S. and Sierra, C. (1997) A framework for argumentation-based negotiation. In: Singh, M., Rao, A. and Wooldridge, M. (eds.), *Intelligent Agents IV, Proc. 4th Int. Workshop on Agent Theories, Architectures, and Languages* (*ATAL'97*): *Lecture Notes in Artificial Intelligence 1365*, pp. 177–192. Springer-Verlag.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Kakas, A. and Riguzzi, F. (1997) Learning with abduction. In: Lavrac, N. and Dzeroski, S. (eds.), *Proc. 7th Int. Workshop on Inductive Logic Programming* (*ILP'97*): *Lecture Notes in Computer Science 1297*, pp. 181–188. Springer-Verlag.

Kakas, A., Kowalski, R. and Toni, F. (1992) Abductive logic programming. *Journal of Logic and Computation*, **2**(6), 719–770.

Kakas, A., Kowalski, R. and Toni, F. (1998) The role of abduction in logic programming. In: Gabbay, D., Hogger, C. and Robinson, J. (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume V: Logic Programming*, pp. 235–324. Oxford University Press.

Kakas, A., Miller, R. and Toni, F. (1999) An argumentation framework of reasoning about actions and change. In: Gelfond, M., Leone, N. and Pfeifer, G. (eds.), *Proc. 5th Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*): *Lecture Notes in Computer Science 1730*, pp. 78–91. Springer-Verlag.

Katsuno, H. and Mendelzon, A. (1991) On the difference between updating a knowledge base and revising it. In: Allen, J., Fikes, R. and Sandewall, E. (eds.), *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR'91*), pp. 387–394. Morgan Kaufmann.

Kifer, M. and Subrahmanian, V. S. (1992) Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, **12**(3–4), 335–367.

Knoblock, C. (1995) Planning, executing, sensing and replanning for information gathering. *Proc. 14th Int. Joint Conf. on Artificial Intelligence* (*IJCAI'95*), pp. 1686–1693. Morgan Kaufmann.

Kowalski, R. and Sadri, F. (1991) Logic programs with exceptions. *New Generation Computing*, **9**(3–4), 387–400.

Kowalski, R. and Sadri, F. (1996) Towards a unified agent architecture that combines rationality with reactivity. In: Pedreschi, D. and Zaniolo, C. (eds.), *Logic in Databases, Proc. Int. Workshop* (*LID'96*): *Lecture Notes in Computer Science 1154*, pp. 137–149. Springer-Verlag.

Kowalski, R. and Sadri, F. (1997) Reconciling the event calculus with the situation calculus. *Journal of Logic Programming*, **31**(1–3), 39–58.

Kowalski, R. and Sadri, F. (1999) From logic programming towards multi-agent systems. *Ann. of Math. and Artificial Intelligence*, **25**(3–4), 391–419.

Kowalski, R. and Sergot, M. (1986) A logic-based calculus of events. *New Generation Computing*, **4**(1), 67–95.

Kowalski, R. and Toni, F. (1995) Reduction of abductive logic programs to normal logic programs. In: Sterling, L. (ed.), *Proc. 12th Int. Conf. on Logic Programming* (*ICLP'95*), pp. 367–381. MIT Press.

Kowalski, R. and Toni, F. (1996) Abstract argumentation. *Artificial Intelligence and Law*, **4**(3–4), 275–296.

Laenens, E. and Vermeir, D. (1990) A fixpoint semantics for ordered logic. *Journal of Logic and Computation*, **1**(2), 159–185.

Lamma, E., Mello, P., Milano, M. and Riguzzi, F. (1999) Integrating induction and abduction in logic programming. *Information Sciences*, **116**(1), 25–54.

Lamma, E., Riguzzi, F. and Pereira, L. (2000) Strategies in combined learning via logic programs. *Machine Learning*, **38**(1–2), 63–87.

Leite, J. and Pereira, L. (1997) Generalizing updates: from models to programs. In: Dix, J., Pereira, L. and Przymusinski, T. (eds.), *Logic Programming and Knowledge Representation, Selected Papers 3rd Int. Workshop* (*LPKR'97*): *Lecture Notes in Artificial Intelligence 1471*, pp. 224–246. Springe-Verlagr.

Levy, A. and Rousset, M. (1996) CARIN: A representation language combining Horn rules

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

and description logics. In: Wahlster, W. (ed.), *Proc. 12th Europ. Conf. on Artificial Intelligence* (*ECAI'96*), pp. 323–327. Wiley.

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

Levy, A., Srivastava, D. and Kirk, T. (1995) Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, **5**(2), 121–143.

Levy, A., Rajaraman, A. and Ordille, J. (1996a) Query-answering algorithms for information agents. In: *Proc. 13th Nat. Conf. on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conf.* (*AAAI/IAAI'96*), **1**, 40–47. AAAI Press/MIT Press.

Levy, A., Rajaraman, A. and Ordille, J. (1996b) Querying heterogeneous information sources using source descriptions. In: Vijayaraman, T., Buchmann, A., Mohan, C. and Sarda, N. (eds.), *Proc. 22th Int. Conf. on Very Large Data Bases* (*VLDB'96*), pp. 251–262. Morgan Kaufmann.

Liberatore, P. and Schaerf, M. (1998) Arbitration (or how to merge knowledge bases). *IEEE Trans. Knowledge and Data Engineering*, **10**(1), 76–90.

Lifschitz, V. (1999) Action languages, answer sets and planning. In: Apt, K., Marek, V., Truszczyński, M. and Warren, D. (eds.), *The Logic Programming Paradigm: A 25-year Perspective*, pp. 357–373. Springer.

Lifschitz, V. and Woo, T. (1992) Answer sets in general nonmonotonic reasoning (preliminary report). *Pages 603–614 of:* Nebel, B., Rich, C., & Swartout, W. (eds), *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR'92*). Morgan Kaufmann.

Lukasiewicz, T. (1998) Probabilistic logic programming. In: Prade, H. (ed.), *Proc. 13th Europ. Conf. on Artificial Intelligence* (*ECAI'98*), pp. 388–392. Wiley.

Lukasiewicz, T. (2001) Probabilistic logic programming with conditional constraints. *ACM Trans. on Computational Logic*. (To appear.) (Also available as: Technical report, INFSYS-TR 1843-00-01, Institute of Information Systems, TU Vienna, 2000.)

MacGregor, R. (1991) Inside the LOOM description classifier. *SIGART Bulletin*, **2**(3), 88–92.

Marek, V. and Truszczyński, M. (1993) *Nonmonotonic Logics – Context-dependent Reasoning*. Springer-Verlag.

Marek, V. and Truszczyński, M. (1994) Revision specifications by means of programs. In: MacNish, C., Pearce, D. and Pereira, L. (eds.), *Proc. Europ. Workshop on Logics in Artificial Intelligence* (*JELIA'94*)*: Lecture Notes in Artificial Intelligence 838*, pp. 122–136. Springer-Verlag.

Marek, V. and Truszczyński, M. (1999) *Logic Programming with Costs*. Unpublished manuscript.

Moore, R. (1985) Semantical considerations on nonmonotonic logics. *Artificial Intelligence*, **25**(1), 75–94.

Nebel, B. (1998) How hard is it to revise a belief base? In: Dubois, D. and Prade, H. (eds.), *Handbook on Defeasible Reasoning and Uncertainty Management Systems: Volume III: Belief change*, pp. 77–145. Kluwer Academic.

Ng, R. (1997) Semantics, consistency, and query processing of empirical deductive databases. *IEEE Trans. Knowledge and Data Engineering*, **9**(1), 32–49.

Ng, R. and Subrahmanian, V. S. (1992) Probabilistic logic programming. *Information and Computation*, **101**(2), 150–201.

Ng, R. and Subrahmanian, V. S. (1993) A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Journal of Automated Reasoning*, **10**(2), 191–235.

Ng, R. and Subrahmanian, V. S. (1994) Stable semantics for probabilistic deductive databases. *Information & Computation*, **110**(1), 42–83.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Niemelä, I., Simons, P. and Soininen, T. (1999) Stable model semantics of weight constraint rules. In: Gelfond, M., Leone, N. and Pfeifer, G. (eds.), *Proc. 5th Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*): *Lecture Notes in Computer Science 1730*, pp. 317–331. Springer-Verlag.

Nute, D. (1994) Defeasible logic. In: Gabbay, D., Hogger, C. and Robinson, J. (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume III*, pp. 353–395. Clarendon Press.

Pearl, J. (1989) Probabilistic semantics for nonmonotonic reasoning: a survey. In: Brachman, R., Levesque, H. and Reiter, R. (eds.), *Proc. 1st Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR'89*), pp. 505–516. Morgan Kaufmann.

Pinto, J. and Reiter, R. (1993) Temporal reasoning in logic programming: a case study for the situation calculus. In: Warren, D. (ed.), *Proc. 10th Int. Conf. on Logic Programming* (*ICLP'93*), pp. 203–221. MIT Press.

Pinto, J. and Reiter, R. (1995) Reasoning about time in the situation calculus. *Ann. of Math. and Artificial Intelligence*, **14**(2–4), 251–268.

Poole, D. (1993) Logic programming, abduction and probability – a top-down anytime algorithm for estimating prior and posterior probabilities. *New Generation Computing*, **11**(3), 377–400.

Pradhan, S. and Minker, J. (1996) Using priorities to combine knowledge bases. *International Journal of Cooperative Intelligent Systems*, **5**(2–3), 333–364.

Prakken, H. and Sartor, G. (1997) Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics*, **7**, 25–75.

Prakken, H. and Sartor, G. (1999) A system for defeasible argumentation, with defeasible priorities. In: Wooldridge, M. and Veloso, M. (eds.), *Artificial Intelligence Today: Recent Trend and Developments: Lecture Notes in Computer Science 160*, pp. 365–379. Springer-Verlag.

Pratt, V. (1990) Action logic and pure induction. In: van Eijck, J. (ed.), *Proc. Europ. Workshop of Logics in Artificial Intelligence* (*JELIA'90*): *Lecture Notes in Computer Science 478*, pp. 97–120. Springer-Verlag.

Przymusinski, T. and Turner, H. (1997) Update by means of inference rules. *Journal of Logic Programming*, **30**(2), 125–143.

Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. and Widom, J. (1995) Querying semistructured heterogeneous information. In: Ling, T., Mendelzon, A. and Vieille, L. (eds.), *Proc. 4th Int. Conf. on Deductive and Object-oriented Databases* (*DOOD'95*): *Lecture Notes in Computer Science 1013*, pp. 319–344. Springer-Verlag.

Ramakrishnan, I., Rao, P., Sagonas, K., Swift, T. and Warren, D. (1995) Efficient tabling mechanisms for logic programs. In: Sterling, L. (ed.), *Proc. 12th Int. Conf. on Logic Programming* (*ICLP'95*), pp. 697–711. MIT Press.

Revesz, P. (1993) On the semantics of theory change: arbitration between old and new information. *Proc. 12th ACM SIGACT SIGMOD-SIGART Symp. on Principles of Database Systems* (*PODS'93*), pp. 71–82. ACM Press.

Rintanen, J. (1998) Lexicographic priorities in default logic. *Artificial Intelligence*, **106**(2), 221–265.

Sadri, F. and Toni, F. (2000) Computational logic and multi-agent systems: a roadmap. *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-net*, 1–31. (URL: `http://www.compulog.org/net/Forum/Supportdocs.html`.)

Sagonas, K., Swift, T. and Warren, D. (1994) XSB as an efficient deductive database engine. In: Snodgrass, R. and Winslett, M. (eds.), *Proc. 1994 ACM SIGMOD Int. Conf. on Management of Data*, pp. 442–453. ACM Press.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press

Sakama, C. (2000) Abductive generalization and specialization. In: Flach, P. and Kakas, A. (eds.), *Abduction and Induction – Essays on their Relation and Integration*, pp. 253–265. Kluwer Academic.

Subrahmanian, V. S. (1987) On the semantics of quantitative logic programs. *Proc. 4th IEEE Symp. on Logic Programming* (*SLP'87*), pp. 173–182. Computer Society Press.

Subrahmanian, V. S. (1994) Amalgamating knowledge bases. *ACM Trans. Database Systems*, **19**(2), 291–331.

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

Thomas, B. (1998) Intelligent Web querying with logic programs. *Proc. Workshop on Inference Mechanisms in Knowledge-based Systems at the Nat. German AI Conf. KI'98*.

Thomas, B. (1999) Logic programs for intelligent Web search. In: Ras, Z. and Skowron, A. (eds.), *Proc. 11th Int. Symp. on Foundations of Intelligent Systems* (*ISMIS'99*)*: Lecture Notes in Computer Science 1609*, pp. 190–198. Springer-Verlag.

Turner, H. (1997) Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, **31**(1–3), 245–298.

van Emden, M. (1986) Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, **3**(1), 37–53.

Wiederhold, G. (1992) Mediators in the architecture of future information systems. *IEEE Computer*, **25**(3), 38–49.

Wooldridge, M. (2000) *Reasoning about Rational Agents*. MIT Press.

https://doi.org/10.1017/S1471068402001576 Published online by Cambridge University Press