INTEGRATING SYMBOL-ORIENTED AND SUB-SYMBOLIC REASONING METHODS INTO HYBRID SYSTEMS

FRANZ J. KURFESS
Computer Science Department, California Polytechnic State University

Knowledge representation and reasoning methods in artificial intelligence almost exclusively rely on *symbol-oriented* methods: Statements describing aspects and objects of the system to be modelled are represented through symbols (mostly text strings), and these symbols are stored in a computer, and manipulated according to the inference rules prescribed by the reasoning method. This works reasonably well in situations where knowledge is available in explicit form, typically through experts or written documents. In situations where knowledge is only available implicitly, e.g. in large data sets, other methods, often based on statistical approaches, have been used more successfully. Many of these methods are based on neural network techniques, which typically represent and process knowledge at a level below symbols; this is often referred to as *sub-symbolic* representation. This contribution discusses approaches to integrate symbol-oriented reasoning methods with sub-symbolic ones into *hybrid systems*.

14.1 INTRODUCTION

This paper consists of three main section: First, we present the foundations by clarifying the terminology used, and by briefly outlining important concepts and aspects of knowledge-based systems as representatives of the symbol-oriented approach, and neural networks as representatives for the sub-symbolic one. Then we discuss general strategies for the combination and integration of the two different types of approaches into variations of hybrid symbol-oriented/sub-symbolic systems, and finally

we present a few specific models for this integration discussed in the literature.

Although artificial intelligence (AI) methods have been applied successfully to a large variety of problems, it has become clear over time that there are substantial fundamental and practical limitations that inhibit their widespread use. Throughout this paper, we will categorize AI approaches into two broad classes: The ones that rely heavily on symbols for the representation and processing of knowledge, and the ones that utilize methods where the role of symbols in knowledge representation and processing is not evident. The first category will be labelled as *symbol-oriented* or *symbolic* approaches, and includes expert systems, theorem provers, or planning systems as typical examples. The second one is referred to as *sub-symbolic approaches*, with neural networks as their main representative. The term sub-symbolic indicates that the basic entity for storage and presentation is at a lower abstraction level than a symbol. These basic entities sometimes correspond to identifiable properties of an object or concept to be represented, and then are often called (micro-) features. At other times, however, the correlation between the item to be represented, and the entities that contribute to its representation in the model is not clear at all.

Large quantities of knowledge and information are nowadays available through computer systems, but our current methods for organization, manipulation, storage, and retrieval are rather tedious. Access to information stored on a computer typically relies either on the knowledge of the location (such as the directory, file name, or Web page), or on syntax-oriented search based on keywords. Whereas more advanced techniques such as the ones used by the Google search engine greatly improve the utility of these approaches, much of the effort in identifying, retrieving, and utilizing knowledge still depends on the human user.

14.1.1 Terminology

The purpose of this section is to clarify the meaning of terms used in the rest of this contribution, in particular the terms *data*, *information*, and *knowledge*. Of particular interest in our context here are also the more specific terms *structured* knowledge, symbol, symbol-oriented, sub-symbolic and symbol grounding.

In this context, the term *data* frequently describes the input and output for computer programs that process these data items in order to provide useful information or knowledge to the human user. An important aspect of data is their rigid, simple, predetermined structure. Typical examples are weather data such as temperature and precipitation collected at various locations and over a certain period of time, or the data collected by credit

card companies for transactions between customers and merchants. It is important to note that data are typically grouped into sets (or records in data base terminology), and that the values stored in such a record can be meaningless without knowledge of the meaning of that particular field within the record.

The term *information* is frequently used as a rather broad and generic term, and – even worse – often as synonym for knowledge or data. It has a precise meaning in some specific areas, such as information theory, but this is not directly applicable to our discussion. The most important aspect for our purpose here is the interpretation of data for human consumption by associating the individual values of related data items with their intended meaning. In the weather example, this is done by converting the set of data (time: April 6, 2002, noon; location: San Luis Obispo; temperature: 23°C; precipitation: 0) into a statement like "Nice weather today at noon in San Luis Obispo". The conversion of data into information typically goes hand in hand with a reduction in the quantity of stored items through an elimination of items that are not usable or irrelevant in a given context.

Although the situation for the term *knowledge* is slightly better than the one for information, there is no clear, widely accepted definition useful for our purposes. The term is typically used to imply a higher level of abstraction than data or information, which again goes together with a reduction in the quantity of stored items. Knowledge usually has a flexible, irregular structure, and is often presented visually as graph with nodes for objects or concepts, and edges for relationships. To emphasize the importance of this aspect of knowledge, we will occasionally use the term *structured knowledge*, highlighting the arbitrary, irregular, and dynamic relationships between individual knowledge items. This is in contrast to relational databases, whose internal structure is expressed through rigidly defined tables that apply to all respective records. To distinguish the irregular nature of the relationships between entities from the regular ones in a database, sometimes the term *semi-structured* is used.

The relationship between data, information, and knowledge can also be visualized as a "knowledge pyramid" with data as the broad foundation at a low level of abstraction, and knowledge as the narrow top with a high level of abstraction. Occasionally *wisdom* is added as an even higher level of abstraction, but for our discussion here this is not so relevant.

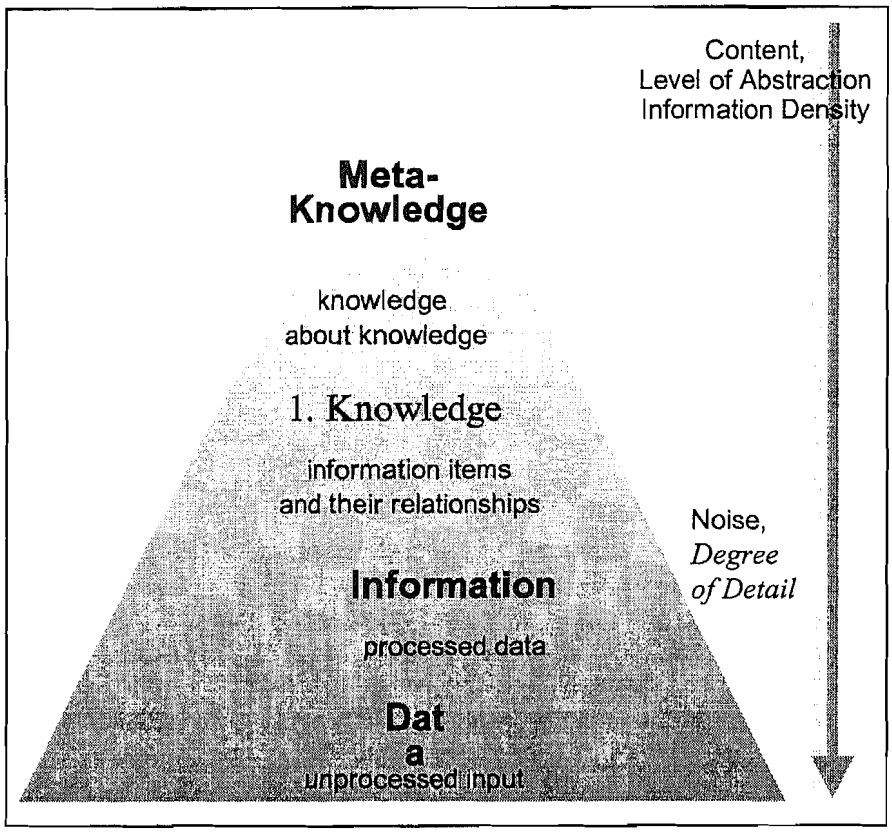

Figure 14.1. Knowledge Pyramid

The table below highlights important aspects of the different knowledge levels from three perspectives: the *real world* as we perceive it, the *computational model* designed to represent certain parts of the real world through a program or computer-based system, and the *abstract model* as captured in a formal specification of the computational model.

With respect to these three perspectives, *data* are usually binary or numerical values obtained through sensors. Initially they are often captured as analogue values, and then digitized for computer processing. *Information* in this context refers to filtered and pre-processed data, selected for their relevance with respect to a particular model. It roughly corresponds to features or properties, and is stored in computational models as the fields of a record or data structure, or the slots and fillers of frame-based representations. This is also referred to as a representation using <attribute, value> pairs. Unless the intended meaning of a value is indicated, e.g. through the name of the attribute, slot, or field, the information is not very useful. *Knowledge* is centred around concepts, which can be viewed as sets

of features that constitute an entity of interest in our model; they may correspond to physical objects in the real world, or mental concepts, and may have an internal structure. Relations capture interesting relationships between concepts. The representation of knowledge in computational models can vary substantially, with objects, rules, data structures and records being among the most frequently used. In abstract models based on mathematical logic, predicates represent the relationships among concepts, and statements are expressed through logical sentences involving logical symbols and terms describing the entities to be represented. *Meta-knowledge* refers to statements about knowledge, and describes how to deal with knowledge. In the real world, methods to capture, store, and retrieve knowledge are meta-knowledge. Thus, a library is a facility utilizing meta-knowledge in order to provide access to knowledge. In computational models, meta-knowledge often is not explicitly represented, but it is evident in the structure of methods, systems, programs, or algorithms dealing with the treatment of knowledge. In the abstract model perspective of mathematical logic, this corresponds to higher order logic, where logical sentences are the actual objects dealt with at higher levels of abstraction.

Table 14.1. Knowledge Levels

Level	Real World	Computational Model	Abstract Model
Meta-Knowledge	statements about knowledge	programs, algorithms	logical theories, inference methods
Knowledge	real-world entities, relations between entities, statements about entities	rules, objects, data structures, records	logical sentences; predicates, logical symbols, terms
Information	identifiable properties of real-world entities	<attribute, value> pairs, slots and fillers, fields	features
Data	analogue sensor readings	(vectors of) digitized sensor values	percepts

14.1.2 Knowledge Representation

One of the critical notions here is the use of *symbols* for knowledge representation. A symbol is a sign or token used to represent an object or concept. It provides unambiguous identification for the specific object, is localized, may have an arbitrary shape, and often needs an explanation in the form of a mapping into a set of terms the user is familiar with. Symbols are usually localized; this means that it is possible to uniquely identify the storage location where the symbol is held. Symbols can be of arbitrary shape, but in connection with computer-based knowledge representation,

strings of characters arranged as words are frequently used. This allows straightforward interpretation by humans. Examples of symbols are π (for the number pi), or the dollar sign $ to indicate that a number is to be interpreted as currency.

In contrast to a symbol, an *icon (*also referred to as *simile*) is a simplified picture that possesses an intended and inherent similarity to the object to be represented. Icons have an obvious mapping between the pictorial representation and the object to be represented. They may be ambiguous, although the context usually provides the necessary information to select the appropriate interpretation. Examples of icons are stylized figures used to mark bicycle paths or parking spaces reserved for disabled people; icons are also frequently used in computer programs, e.g. by employing the stylized image of a printer to indicate the printing function.

An *index* is an indicator that elicits an important feature of an object or concept. It is usually unambiguous, uses a localized representation, and provides an inherent mapping from the representation used to the feature to be represented. Examples are a thermometer (for the temperature), a battery or gasoline gauge to indicate the status of a battery or a gas tank in a car, or a clock to indicate time.

14.2 SYMBOLS AND FORMAL SYSTEMS

Symbols, in the form of human-readable names for variables, constants, methods, objects or other entities used for knowledge representation, are very important for the use of computers to store and process knowledge. They are extremely helpful for the design and construction of computational models that capture and simulate relevant aspects of a system. In order to relate such a computational model to the real world, a semantic for the formal system is needed to help with the interpretation of observations of the model. Sometimes symbols are defined only in reference to other symbol structures, which often allows for the elegant design and implementation of a complex model, but can have the danger of being detached from the real system for which it is supposed to stand. The semantics should be based on concrete experiences with a real environment, not only in reference to other symbol structures. The meaning of symbols is often assigned by the programmer or designer, and usually relies on the selection of names based on words that convey the intended interpretation of the aspect or concept the symbol stands for. Whereas this is a very practical strategy to help with the interpretation of the model, it relies on a "parasitic" mapping, where the association between the symbol and the corresponding aspect of the real world are not intrinsic, but dependent on the interpretation of the chosen

name. This opens up the possibility of errors due to the interpretation of symbols based on the meaning of the words chosen in the string representation, rather than the specific concept or aspect they are intended for. And of course, unless the language or terminology used is familiar to the user of the symbol, an interpretation may become very difficult or impossible. This is especially important for computer-based knowledge processing: since computers do not perform the implicit interpretations we humans do automatically when we read these symbols, the meaning intended by the programmer is not accessible to the computer.

Most computer-based systems for the representation and processing of knowledge rely on *symbolic representation* and the corresponding symbol-oriented access and manipulation methods. Expert systems or more formal, logic-based representation schemes clearly fall into this category. As an alternative, so-called *sub-symbolic representation* principles rely on the representation of objects or concepts through (micro-) features. Instead of representing an entity through a direct mapping onto a unique symbol, an entity is represented through a set of features that uniquely identify the entity. Representational aspects of knowledge items are then accessible at a level below symbols, i.e. through features (which capture relevant aspects of the entity) or micro-features (which capture aspects that by themselves are not particularly meaningful, only in combination with other micro-features). These sub-symbolic methods are frequently used in combination with *distributed representation* schemes, where an entity is not mapped to a specific location in memory, but distributed over several locations. Such schemes are most frequently used in neural networks, where individual neurons contribute to the representation of multiple entities, and an individual entity's representation is distributed over several neurons. Distributed representation enables more flexible access methods based on the similarity of entities as expressed through overlaps in the set of features that describes them. On the other hand, it causes problems with symbol manipulation as knowledge processing mechanism, and it is difficult to develop sound and efficient methods for the processing of knowledge.

14.2.1 Symbol Grounding

The establishment of a mapping between the symbol and the object or concept it is supposed to represent is sometimes called "symbol grounding," implying a close relationship between the symbol and the corresponding entity in the real world. In symbol-oriented systems, this relies on human interpretation, often based on the "parasitic" mapping through strings of characters that have meaning for humans. Unless computers have an understanding of the words used in these mappings, it is not sufficient for

machine-based interpretation. For example, in the design of maps used by robots for navigation purposes, designations such as "bathroom-door" may be very useful for humans, but of little use to a robot. Symbol grounding is often employed in the context of *emergent* grounding, where the mapping between the real world and its representation is not imposed by a designer or programmer, but rather arises on its own while the representation for a specific entity in the real world is constructed. The mapping is constructed together with the acquisition of the respective data or knowledge, and ideally should provide a causal correlation between the data reflecting the status of the environment and the internal representation of the respective aspects in the model.

14.2.2 Knowledge Representation Formalisms

Frameworks to describe knowledge items and their relationships, ideally with formal underpinnings suitable for some theoretical treatment, provide the foundation for the representation manipulation of knowledge via computers. The formal treatment should enable proofs of critical properties, the determination of time and space complexity, and other important aspects as indicated in the table below. In addition to the formal perspective, the translation of a formalism into a practical system of course must also be taken into consideration.

Table 14.2. Knowledge Representation Criteria

Criterion	Issues
adequate	are essential aspects captured?
comprehensible	is the represented knowledge understandable?
transferable	can the knowledge be communicated?
uniform	is identical information consolidated?
composite	can components be grouped into ensembles?
reliable	belief / truth, consistency
verifiable	objective / subjective, facts, derived knowledge, basic assumptions
efficient	usage of space, execution time for basic operations

Traditional knowledge-based systems are almost exclusively based on symbolic knowledge representation and manipulation methods. The availability of expertise, explicit representation of knowledge, ease of modification, consistency of answers, and the accessibility of the knowledge are important practical considerations for the design and realization of such systems. Among their potential disadvantages are limited knowledge (in particular the lack of "common-sense" knowledge), the treatment of

incomplete or inexact data, possibly incorrect answers, low comprehensibility, and brittleness.

Some of these problematic aspects, however, are strong points of alternative computational paradigms such as fuzzy logic and neural networks. The next section gives a very short overview on neural networks and their use for the representation and processing of knowledge.

14.2.3 Basic Concepts: Neural Networks

In most applications of neural networks, they are used for processing of elementary data items at a relatively low level in the knowledge pyramid. Many popular types of neural networks take vectors or other simple, very regular data structures as input, and produce again relatively simple data structures as output. Information is stored implicitly through parameters of the network, most frequently: interconnection weights, and processing of information is achieved through propagation of activities in the network. The main activities in a neural network are the storage of information, often through "learning," and the recall of the stored information. Usually there is no explicit generation of new knowledge, although some learning and recall activities include operations like generalization, or recall of similar items if there is no exact match. The learning capabilities of neural networks are often applied to sets of sample data, which the network can use to generate an internal representation that allows it to select the most suitable response for new data.

In the following, we will very briefly review a generic model for artificial neural networks, and then discuss interesting aspects of some types of neural networks, particularly with respect to the overall theme of representing and processing knowledge.

14.2.4 Artificial Neural Network

An *artificial neural network (ANN)* can be viewed as a collection of neuron-like computational elements with weighted connections between the elements The nodes perform simple functions like addition, multiplication, or threshold comparison, while the weighted connections store information. This storage of information is achieved by learning through the adaptation of weights in reaction to the presentation of sample data. An individual neuron receives input either from outside the network, or from other neurons via interconnections. It sums up the weighted inputs affiliated with incoming connections, applies the activation function (e.g. threshold or some other nonlinear function), and then generates a response propagated through the output. Neurons typically have multiple inputs with positive (excitatory) or

negative (inhibitory) weights, but calculate only a single output (which can be propagated to several other neurons, however).

Within the context of storing and processing knowledge, two approaches are most frequently used for neural networks: Local representation and distributed representation. A scheme in which an object or concept is represented by one single neuron is considered a *local representation*, where conceptual entities correspond to individual neurons, and each neuron is affiliated with the representation of only one entity. Relationships can be directly expressed by connections between individual neurons, and a neural network essentially becomes an implementation vehicle for a semantic network, or some similar, graph-based representation. A localist representation is relatively explicit and easy to understand, and can be generated from other representation methods via systematic transformation or compilation. Knowledge processing on a localist basis becomes more complicated since the variety of relations between the nodes requires corresponding inference rules to combine the individual pieces of knowledge into new ones. Most localist schemes also have difficulties with learning algorithms; in principle, the links between nodes can be associated with weights, but the encoding of sample data in such a pre-configured network is difficult to achieve.

In a *distributed representation* scheme, an entity is represented jointly by several neurons, and each neuron contributes to the representation of several entities. The representation of relationships between entities is more complicated in this scheme; in principle, it can also be achieved through a distribution of the links, but this entails a distributed reasoning method for knowledge processing. Networks using a distributed scheme are usually constructed implicitly through learning, rather than explicitly through compilation. Learning in this case is much easier since the distributed nature of the network allows for a greater degree of freedom, thus facilitating the formulation and use of learning algorithms.

14.2.5 Feedforward Networks

In one of the most popular network configurations, the *multi-layer feedforward network*, nodes of the network are arranged in a small number of layers, typically two to four. Nodes are only connected to nodes in the next layer, not within the same layer. Frequently used interconnection patterns between layers are fully connected, where each node in one layer is connected to all nodes in the next layer, one-to-one connections between nodes in adjacent layers, and partially connected patterns. The flow of activity through the directed links is from the input layer through the hidden layers to the output layer, and the corresponding interconnection graph

contains no cycles. This limits the capabilities of such networks, but makes their behaviour computationally manageable, with a guaranteed response time between the application of an input pattern and the response by the network.

Feedforward networks are often used in combination with the *backpropagation learning algorithm*: For each pair of input pattern and desired response, the network calculates the current response according to its configuration of weights, and compares the result against the desired response. The difference between the two is used to adjust the weights between the output layer's nodes, and those of the layer next to it. This can be taken as an indication of the desired response at that layer, and be applied in the same way to the weights between that layer and the previous one, and so on. Although this algorithm can be time-consuming and may require adjustments of some parameters, it often results in a network that can produce sensible responses to input patterns close enough to the set of samples. Feedforward networks are suitable for representing mappings between sets of individual pairs of input patterns and desired output patterns.

14.2.6 Recurrent Networks

In their most frequent incarnation, *recurrent networks* are organized in layers, just like feedforward networks. In addition to the interconnections directed from the layers close to the input towards the output layers, recurrent networks also have connections going the other way, e.g. from the output layer to the closest hidden layer, or from one hidden layer to a previous one, or to the input layer. Due to their more complex internal structure, recurrent networks are capable of more sophisticated internal representation, at the expense of more complicated learning methods, and retrieval behaviour that is difficult to analyse. Recurrent networks are capable of capturing relationships between individual input patterns, and can learn the mapping of sequences of input patterns into sequences of output patterns. Methods have also been devised to represent graphs in recurrent networks.

14.2.7 Knowledge Representation and Neural Networks

Since neural networks typically are used for lower-level data processing rather than knowledge representation and manipulation tasks, this section will examine important advantages and problems of neural networks for such tasks. One of the very basic problems of knowledge representation is to store large sets of features or sample data. In conventional systems, this is frequently done through vectors, arrays, or records in databases. This is also

fairly easy with neural networks through the use of vectors; for more complex records, it becomes more complicated. Neural networks have two important advantages over conventional methods: They typically have a very quick, fixed response time, and are capable of generalization: if no vector can be found that exactly matches the given input, the closest one is automatically chosen. Another very basic task in knowledge representation is relating features to specific entities, e.g. by associating descriptors like name, height, hair colour, eye colour, etc. to individual persons. In conventional systems, this can be achieved fairly easily through records, objects, terms or other representation methods that provide internal structure to entities. Processing such information and knowledge becomes more difficult, essentially requiring variable binding and unification for more complex tasks. Neural networks capable of capturing this internal structure, and performing appropriate operations on the stored structures, have been around for quite a while, frequently under the term *connectionist networks*. The guiding principle behind them is to map the structure of an entity to be represented onto a set of nodes with appropriate connections, effectively mirroring the original structure in the network. Processing is then performed by the propagation of activation through the network. Such networks can emulate many of the operations performed by conventional, symbol-oriented approaches, and have some advantages due to their massively parallel mode of operation. On the other hand, the typical symbol manipulation operations would require a reconfiguration of the network on the fly, and the development of learning algorithms for such networks is a major challenge. From a graph-based perspective, many knowledge representation and manipulation tasks can be viewed as the mapping of a general graph onto a network with a fixed topology and size. Storing and retrieving the graph, together with operations on the stored structures are the main challenges here. Some approaches have been devised, mainly based on recurrent networks, to fold graphs into a network and subsequently unfold them; these efforts are still in an experimental stage. If successful, however, they offer very interesting application such as fast retrieval based on structural similarity, similarity-based graph matching, or "sloppy" unification of complex structures.

14.3 INTEGRATION OF SYMBOL-ORIENTED AND SUB-SYMBOLIC SYSTEMS

A combination of different approaches to knowledge representation and processing, with expert systems as example of symbol-oriented system and neural networks as examples of sub-symbolic systems, appears very

promising due to the duality of the approaches: Symbol-oriented systems proceed in a methodical, precise, formal, but sometimes brittle and slow manner, whereas neural networks are faster, generality- and similarity-oriented, and employ relatively robust, but not necessarily precise operations. In the following we will discuss some approaches to the integration of symbol-oriented with sub-symbolic knowledge representation and processing methods: stand-alone, transformational, loose coupling, tight coupling, and full integration.

14.3.1 Stand-Alone

Independent components based on different methods are at the core of this approach. The use of pre-existing components, either in software or hardware, offers a simple implementation, especially in the most extreme case with no direct interaction between the components. Redundancy to provide a backup in case of failure, validation where one component is used to confirm the other's results, or the utilization of prototypes as quick proof of the conceptual approach are reasons to choose this integration method. It profits from the different capabilities, such as learning and generalization for neural networks, and deduction and explanation for expert systems. Although it may be considered a degenerate case of integration, obvious benefits like simplicity, ease of development, independence, and redundancy can overcome the limitations, which include the lack of transfer of information between the components, multiple maintenance (especially if the same knowledge is represented in multiple components), no mutual balance of the underlying methods, and the possible lack of consistency.

14.3.2 Transformational

The *transformational approach* utilizes the conversion between a conventional representation scheme, such as the rules of an expert system, or the graph of a semantic network, to a neural network and vice versa. This transformation must maintain the essential properties of the source representation in the target representation. The conversion of a collection of facts and rules in the knowledge base of an expert system into a neural network establishes prior knowledge in the network. This can help making the learning task easier, and is often used to fine-tune the rules and facts in a knowledge base with a set of samples representative for the domain under consideration. The transformation into a neural network can also offer advantages in the response time of the system, the adaptability through the learning algorithm of the network, and higher robustness due to its generalization capabilities.

The transformation of a neural network into a set of rules and facts can be used to generate a more explicit, symbol-oriented representation, and is usually referred to as *rule extraction*. This is appropriate when a collection of sample cases is available to be used for the training of a neural network, but a more explicit representation is desirable, e.g. for reasoning or explanation purposes. The idea here is to maintain the learning and generalization capabilities of the neural network, while also employing the higher-level manipulation methods of symbol-oriented knowledge representation schemes. It can be applied to data-intensive problems, where neural networks serve as the first filtering and generalization step, but a more explicit representation is required for the documentation and verification of knowledge. It is also used as an analysis tool for neural networks, providing a justification and explanation of their hidden contents via the translation into rules that are more amenable to human inspection.

Systems based on a transformational approach can usually be developed quickly, assuming that the source and target representations are already used in a component of the system. What is needed then is only a transformation from one representation to another one. In comparison with two standalone systems, knowledge maintenance is necessary only for one system, although the two components implementing the neural network and the symbol-oriented representation themselves still remain. A transformation-based system also offers a choice of development as well as operation: Depending on the most critical factors, knowledge acquisition can be performed via learning from samples by the network, or the formulation of rules by humans, and the system can operate based on fast and robust responses from the neural network, or on the methodical, explicit, but often much slower reasoning from the symbol-oriented component.

The methods available for the transformation between neural networks and symbol-oriented approaches are still in their infancy, and no fully automated transformations applicable to general problems are available. It is often necessary to develop specific approaches for new domains or major modifications to a system. The conservation of equivalence for the transformation between such different knowledge representation and manipulation methods is a major fundamental problem, especially when combined with operational limitations.

14.3.3 Loose Coupling

In the transformational approach, the whole body of knowledge represented in the system is converted from one knowledge representation scheme into another. In many circumstances, this approach may not be appropriate or impractical, and the exchange of smaller pieces of knowledge

between specific components may be more desirable. This is often referred to as *loose coupling* between components; communication via files, pre-/post-processing, and the use of front or back ends for special tasks are practical examples. This can be done in a sequential way, where one component's output constitutes the input for the next one, or through co-processing, where several components are active simultaneously and exchange information when needed. This interaction and cooperation can be applied to data refinement, problem solving, or decision-making. Another domain is user interface design, with the goal of more flexible user interactions through speech processing, handwriting recognition, or user modelling.

Similarly to transformational approaches, loosely coupled systems are often easy to develop since they tend to rely on existing components, with a relatively straightforward system design and implementation. Additional work is needed to establish a protocol for the exchange of knowledge between components, and to synchronize the activities of the individual components. Loosely coupled systems may also exhibit some redundancy across their components, and can incur high communication costs.

14.3.4 Tight Coupling

Instead of exchanging individual knowledge items through message passing or similar mechanisms, *tightly coupled* systems establish communication via shared memory. In this scheme, memory-resident data structures of one component are directly accessible to other components, allowing quick interaction between components. With respect to knowledge processing, such systems are often referred to as blackboard architectures with communication via shared data structures stored in a commonly accessible memory area, the blackboard. In this approach, components exchange information directly. It is used for independent components or agents that constitute cooperating systems, or for embedded systems where components of one kind are embedded inside a system of another kind.

Tightly coupled systems often offer great design flexibility and robust integration while achieving a reduced communication overhead, thus leading to higher performance than loosely coupled systems. Sometimes it is possible to develop a system that conceptually uses loose coupling at the design level, but implements the exchange of information through shared memory for performance reasons. On the other hand, they typically have an increased complexity with higher interdependence among the components, and are more difficult to develop.

14.3.5 Full Integration

In all of the above approaches, the different components of the system may use their own internal representation scheme, and they exchange knowledge in different ways. A *fully integrated* system relies on a shared knowledge representation for all of its components. It often exhibits a dual nature, enabling symbolic and sub-symbolic interpretation of represented items, and the corresponding operations for storage, retrieval, and manipulation. In such a system, communication is performed implicitly via a shared representation mechanism. From a knowledge representation perspective, there may be no separate components for the storage, manipulation, and retrieval of knowledge, and the distinction between symbol-oriented and sub-symbolic methods becomes superfluous.

Such fully integrated systems often have increased capabilities in comparison with the other approaches. There is no redundancy due to replication of features or functions, and in principle, high performance is achievable through the efficient shared representation. These systems are also prone to high complexity, exaggerated by the lack of methods and tools for the design, implementation, validation and verification. The lack of redundancy may also cause lower fault tolerance.

14.3.6 ES + NN Hybrids

The use of hybrid systems at this time is still rather limited, and mostly constrained to research and experimental settings. The most frequently used configuration is an architecture with expert system and neural network components, and the transformational technique of rule extraction as the basis of knowledge sharing between the components. Such systems offer the mutual benefits of enhanced capabilities and operational characteristics, usually with the goal of better performance and higher fault tolerance. Thus, systems can be designed that combine the strengths of the two approaches, while their deficiencies are overcome by the other technique.

14.4 CONCLUDING REMARKS AND OUTLOOK

Expert systems offer a comprehensible knowledge representation, with tools and methods for explanation to humans, and formal methods for validation and verification. Their separation of knowledge and inference engine makes modifications of the stored knowledge relatively easy, although consistency and coherence can become problematic for large collections. Commercial tools are available for development and implementation, and a reasonably

large body of experience has been established. On the other hand, knowledge acquisition often constitutes a bottleneck for expert systems, and may require the use of domain experts and knowledge engineers with high costs and limited availability. In addition to the problems associated with the complexity of large systems in general, expert systems also have difficulties with common-sense knowledge, learning, and brittleness.

Neural networks can be helpful with knowledge acquisition due to their capability to learn from examples. In some situations, their generalization capability allows the design of more robust systems, assuming that the set of samples presented to the network for learning is representative for the application problem as a whole. For many types of networks, the generation of a response to a query requires one sweep of activation from the input layer through hidden layers to the output layer, leading to a very short, constant response time. This can be a major performance advantage over a rule-based system with its deliberate reasoning with an indeterminate response time. Neural networks suffer from an incomprehensible representation, requiring elaborate analysis methods and visualization tools to offer some insight into their internal representation of knowledge. To obtain an explanation of why a particular response was generated is even more difficult. The most commonly used types of neural networks have very limited reasoning capabilities, essentially restricted to generalization. Although elaborate inference mechanisms can be constructed from neural components, they are not commonly used at this time.

The synergy between expert systems as representatives of symbol-oriented and neural networks as representatives of sub-symbolic approaches relies on their complementary features: Expert systems work from a logical, symbolic, explicit basis, while neural networks rely on numeric, associative, and implicit operations. One frequently used approach is to enhance knowledge acquisition for expert systems through neural networks, e.g. by using collections of samples representing the problem domain. This is also used for the modification of knowledge through the adaptation of sets of facts and rules to a statistical basis. A collection of facts and rules established with the help of an expert, for example, can be augmented by combining it with a neural network that was trained with a representative set of actual problem cases from the same domain. The same techniques of rule extraction are sometimes used for the investigation and explanation of the internal representation of knowledge in neural networks. Other approaches have used neural networks to learn heuristics for problem solving, essentially enabling learning from experience. In this case, important aspects of decisions during the problem-solving process are learned by a neural network, and applied when a similar situation occurs. In the other direction,

prior knowledge in the form or rules can be used for "priming" neural networks, leading to faster learning and better generalization.

Many problems could benefit from the combination of symbol-oriented and sub-symbolic methods for representation and processing of knowledge. Since the two approaches are complementary with respect to their computational properties, the design and development of hybrid systems combining both seems promising from a conceptual perspective. To date, most applications of such hybrid systems are still experimental in their nature, but there is a growing interest outside the research community. Depending on the underlying representational structures, a loose or tight coupling between the individual components can be achieved. Examples of domains and applications where hybrid systems seem suitable are molecular biology, retrieval and organization of structured documents (such as texts, drawings and diagrams), or component-based software systems. Respective examples of entities to be represented and manipulated are molecular structures, hyperlinked documents containing knowledge in various formats, handwritten characters or natural language constructs.

It is clear that substantial work is needed before hybrid systems will be widely used in practical applications. Whereas symbol-oriented methods and technologies have been in use for quite a while, mainly in the form of expert systems, the use of neural networks for knowledge-related tasks does not have a rich history. Thus, in addition to the technical problems of integrating different components, some fundamental methods to improve the knowledge representation and manipulation capabilities of neural networks must be investigated more deeply. This includes the evaluation of different approaches to represent and process structured knowledge with neural networks, especially concerning expressiveness, complexity, learning methods, and performance. Such networks should be capable of representing general graphs, with (approximate) graph matching as one of the very essential manipulation methods. From a more practical perspective, the identification and selection of candidates for a test suite enabling meaningful comparisons between different approaches is very important.