CAKE: A Computer Aided Knowledge Engineering Technique

Patrick Doherty[1] and **Witold Łukaszewicz**
Andrzej Szałas[2]

1 Introduction

Logic engineering often involves the development of modeling tools and inference mechanisms (both standard and non-standard) which are targeted for use in practical applications where expressiveness in representation must be traded off for efficiency in use. Some representative examples of such applications would be the structuring and querying of knowledge on the semantic web, or the representation and querying of epistemic states used with softbots, robots or smart devices. In these application areas, declarative representations of knowledge enhance the functionality of such systems and also provide a basis for insuring the pragmatic properties of modularity and incremental composition. In addition, the mechanisms developed should be tractable, but at the same time, expressive enough to represent such aspects as default reasoning, or approximate or incomplete representations of the environments in which the entities in question are embedded or used, be they virtual or actual.

Equally important are the tools used to do the modeling. Although difficult to evaluate formally, such modeling tools should provide straightforward methods which insure the modularity and incremental composition of the knowledge structures being designed in addition to guaranteeing formal semantics and transparency of usage.

The applications we are involved in require an efficient representation and query mechanism for the knowledge structures and epistemic states used by robots or softbots, in particular for applications where planning in the context of incomplete states and approximate knowledge is a necessity. We have focused on a generalization of deductive databases and query languages which involves the use of rough knowledge databases (databases where approximate relations and properties are the rule rather than the exception) and where queries can be non-monotonically contextualized to locally close only parts of the database since a closed-world assumption is not feasible. This approach provides us with a reasonably efficient query mechanism and a reasonably expressive query language for querying approximate epistemic states. In such knowledge structures, both positive and negative knowledge must be stored explicitly to ensure an open-world assumption.

In the approach we are pursuing, we view a (generalized) database as a loosely coupled confederation of granular agents, where each agent is responsible for managing all or part of a relation or property. In fact, several agents may contribute locally to the definition of a relation. In addition, each relation is viewed as a partial or approximate object represented in terms of positive and negative information. Granular agents may be composed and abstractions of these compositions (called modules) can be constructed where the module is viewed externally as the manager of a specific relation, hiding the complexity of generating its extension. Modules may be defined recursively in terms of other modules or as combinations of modules and explicit types of granular agents.

Querying such confederations of active knowledge can be done in a number of ways using a number of querying techniques. For instance, certain granular agents may manage and compute default rules, while others may adjudicate between several default agents when there is conflict. Other agents may manage a local context which locally closes or minimizes, maximizes or fixes several different relations.

These mechanisms are intended to be used in environments where knowledge or information is distributed, often times locally inconsistent, and where granular agents can compose and decompose dynamically in order to represent knowledge structures and query them in a flexible and tractable manner. In order to construct such knowledge structures and granular agent confederations in a principled and straightforward manner, we propose a diagrammatic technique for building representations and doing inference which insures formal correctness. The diagrammatic technique and its semantics will be the focus of this article.

We call the method CAKE, an acronym which stands for *computer aided knowledge engineering* and which encompasses the techniques and functionalities already described informally in addition to others reported elsewhere. CAKE provides us with a means for constructing and visualizing the complex dependencies between granular agents. It can be naturally viewed as an extension of well-known entity-relationship diagrams designed for representing relations in relational databases. It also provides tools to represent a complex querying mechanism for generalized deductive databases, which is expressive enough to model numerous knowledge representation paradigms, including defaults and many circumscription policies.

CAKE enjoys two important properties. Firstly, it has a simple well-defined semantics. Secondly, it is tractable: any reasoning process that can be represented using CAKE is computable in polynomial time. This makes our formalism attractive from the standpoint of practical applications.

This article is organized as follows. We start with a brief presentation of the basic concepts associated with the CAKE method and illustrate our formalism with a few examples concerning default logic.[3] Section 3 provides a semantics and computational mechanism

[1] Department of Computer Science, Linköping University, S-581 83 Linköping, Sweden
[2] College of Economics and Computer Science, TWP, Wyzwolenia 30, 10-106 Olsztyn, Poland

[3] We assume that the reader is familiar with standard default logic, see [3, 4]. We choose these examples to demonstrate continuity with previous work

for CAKE. In section 4, we show how CAKE's computational mechanism works in practice. Finally, section 5 contains concluding remarks.

2 Introduction to CAKE

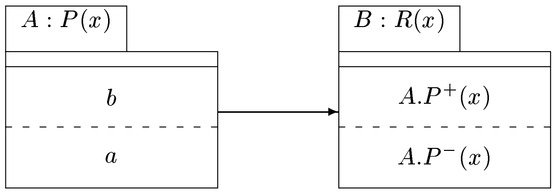
CAKE is heavily influenced by the notion of a *rough relation*. Such a relation R is understood as a triple of relations $\langle R^+, R^-, R^\pm \rangle$, where R^+ represents those tuples of objects of the considered domain that are known to satisfy R, R^- represents those tuples of objects that are known to satisfy the complement of R and $R^\pm$ represents the remaining tuples. In what follows R^+, R^- and $R^\pm$ are called the *positive, negative* and *boundary part (region)* of relation R, respectively. In applications, many of the relations used can be induced via supervised learning techniques and stored as rough relations in a database.

The CAKE method is based on the use of three types of diagrams representing granular agents, adjudicating agents and knowledge modules, respectively.

The basic concept of the CAKE method is that of a *granular agent*. Such an agent is to be thought of as an abstract object storing information about a rough relation. Each agent is responsible for delivering a single relation, though a relation can be distributed among many agents.

An agent can store its own facts, as well as rules defining a relation or imposing some constraints on it. The rules define a computational mechanism which allows the agent to compute the relation. Such a mechanism is called an *agent method*. It should be emphasized that an agent method may refer to information stored by other agents. In fact, there can be considerable dependencies among agents. In order to make the references to relations unambiguous, we use notation $A.R$ to indicate that relation R comes from agent A.

Each granular agent is graphically represented by a diagram. Below, we show two diagrams corresponding to two granular agents.



The left diagram represents a granular agent A responsible for a unary relation P. This is denoted at the top part of the diagram by the *label* "$A : P(x)$". The rest of the diagram is horizontally divided into three parts, referred to (from top to bottom) as a *context part*, a *negative part* and a *positive part* of the diagram, respectively.[4] In the case of agent A, the positive (resp. negative) part of the diagram contains individual constants that are known to satisfy the relation P (resp. the complement of P). Accordingly, the information stored by agent A consists of two facts: $P^+(a)$ and $P^-(b)$.

The right diagram represents a granular agent B responsible for a unary relation R. The negative and positive parts of the diagram represent methods the agent uses. The method $A.P^+$, placed in the negative part of the diagram, is interpreted as the following rule: "for any object x, if x is in the positive part of the diagram of agent A, infer that x is in the negative part of the diagram of agent B". Similarly, the second method represents the rule: "for any object x, if x is in the negative part of the diagram of agent A, infer that x is in the positive part of the diagram of agent B". Given these methods, agent B can infer $R^-(a)$ and $R^+(b)$.

Notice that there is an arrow from A to B. This represents the fact that agent B uses information stored by agent A. There is a dependency between agent A and B.[5]

An important concept used in the CAKE method is that of a *knowledge module*. Such a module may be viewed as a complex agent consisting of a group of granular agents.[6] One of the main roles of a module is to deal with contradictions. The next example will help us illustrate this.

Example 2.1 Consider the default theory[7]

$$T = \langle \{Q(n), R(n)\}, \frac{R(x) : \neg P(x)}{\neg P(x)}, \frac{Q(x) : P(x)}{P(x)}\} \rangle.$$

This is the "Nixon diamond" theory with R, P, Q, n standing for "Republican", "Pacifist", "Quaker" and "Nixon", respectively.

Figure 1 represents the diagram corresponding to T.

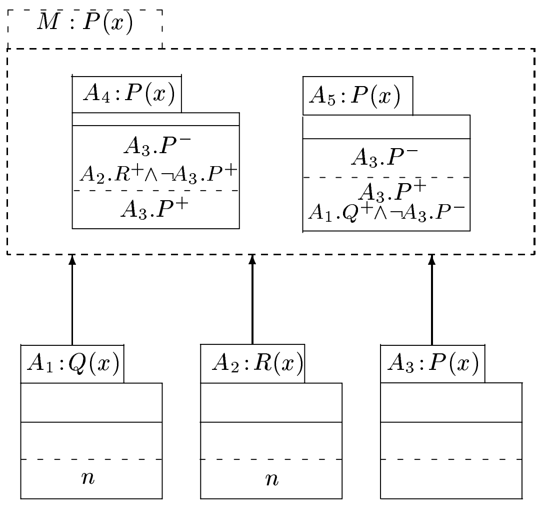


Figure 1. Diagram corresponding to the theory of Example 2.1.

The agents, A_1, A_2 and A_3 at the bottom of the diagram are granular agents representing information extracted from the axioms of the theory. Since n is in the positive part of the diagrams of both A_1 and A_2, the facts stored by these agents are $Q^+(n)$ and $R^+(n)$, respectively. Both the negative and positive part of the diagram of agent A_3 is empty, because no specific information concerning the relation P

and obviously not for their novelty!

[4] Due to lack of space, only a fragment of the CAKE method will be presented here. In particular, the context part of a diagram will not be used in the examples considered in this paper. This part contains rules allowing one to deal with local closed-world assumption policies which are often very useful in practical applications, in particular planning in the context of an open-world assumption (see [2]).

[5] Observe that relation P associated with agent A is denoted by $A.P$ in the diagram of agent B. This is because many agents may be responsible for a single relation.

[6] CAKE also permits modules containing other modules, but this possibility will not be considered in the article.

[7] Recall that $\frac{A(\bar{x}):B(\bar{y})}{C(\bar{z})}$ denotes a default rule, where formulas $A(\bar{x})$, $B(\bar{y})$ and $C(\bar{z})$ are called a *prerequisite*, a *justification* and a *consequent* of the rule, respectively. The intended meaning of the rule is "if $A(\bar{x})$ holds and $B(\bar{y})$ is consistent with the current knowledge then by default assume that $C(\bar{z})$ holds, too".

can be extracted from the axioms of T. The agents A_4 and A_5 manage the defaults of the theory. They are responsible for the relation P which occurs in the consequent of both of the defaults. There are three methods associated with each of these agents. Two of them, namely $A_3.P^-$ and $A_3.P^+$, allow agents A_4 and A_5 to use knowledge of agent A_3 while computing the relation P. Using them, the agents can infer that, for any object x, x lies in the negative (resp. positive) parts of their diagrams, provided that x lies in the negative (resp. positive) part of the diagram of A_3.

The remaining methods, namely $A_2.R^+ \wedge \neg A_3.P^+$ and $A_1Q^+ \wedge \neg A_3.P^-$, represent defaults in the theory. Observe that method $A_2.R^+ \wedge \neg A_3.P^+$, used by agent A_4, is placed in the negative part of its diagram. Accordingly, it has the following meaning: "for any object x, if x is in the positive part of the diagram of A_2 and it is not the case that x is in the positive part of the diagram of A_3, infer that x is in the negative part of the diagram of A_4. Similarly, the meaning of method $A_1Q^+ \wedge \neg A_3.P^-$, used by agent A_5, is the following: "for any object x, if x is in the positive part of the diagram of A_1 and it is not the case that x is in the negative part of the diagram of A_3, infer that x is in the positive part of the diagram of A_5. Since both A_4 and A_5 make inferences concerning the relation P, they have been grouped into a single knowledge module, labeled $M : P(x)$. It is the module, not an individual agent, that is responsible for default inferences about P, although the individual agents in the module contribute to its semantics.

Suppose the query we (or an agent) are interested in is $P(n)$. The answers of agents A_4 and A_5, obtained by their default methods, are clearly False and True, respectively. This conflict is resolved by module M by means of CAKE's standard voting mechanism which is implicitly used by default if no explicit voting method is provided. It is based on the principle that whenever a query is answered True and False within a knowledge module, the answer output by the module is Unknown.[8] ∎

Although the standard voting mechanism used by CAKE is very natural from an information ordering perspective, it is often reasonable to use other voting policies. The CAKE method allows the user to define her/his own voting policy. This is technically done by introducing a special agent, called an *adjudicating agent*, as part of a module where the agent is responsible for managing the voting process when required by the module.[9]

Example 2.2 Consider a well-known default theory T given by

$$W = \{A(j) \wedge FTS(j)\}$$
$$D = \left\{ \frac{FTS(x) : \neg E(x)}{\neg E(x)},\ \frac{A(x) : E(x)}{E(x)} \right\}$$

where A, FTS, E and j stand for "Adult", "FullTimeStudent", "Employed" and "John", respectively.

¿From the syntactic point of view, the theory T is identical with that from Example 2.1. Accordingly, if we represented T by a diagram analogous to that previously used, the answer to the query $E(j)$ would be Unknown. Obviously, this answer is intuitively problematic. Given that John is both a full time student and an adult, the answer to the query should be False if a concept specificity criterium is used to adjudicate conflict in a hierarchy. To achieve this effect, we have to replace the implicit default voting policy with another. This policy will give a higher priority to the first default of the theory. The appropriate diagram corresponding to T is presented in Figure 2.

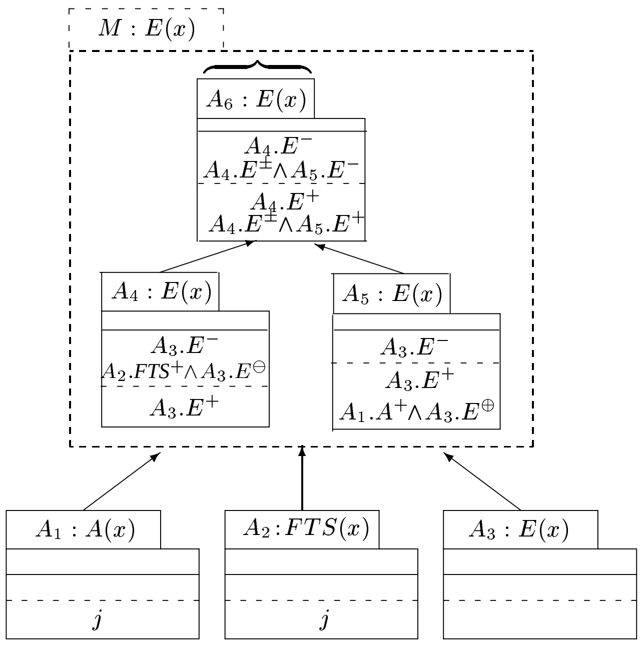


Figure 2. Diagram corresponding to the theory of Example 2.2.

Agent A_6 is an adjudicating agent.[10] The methods $A_4.E^-$ and $A_4.E^+$, employed by the agent, allow it to use the knowledge of agent A_4 while computing the relation E. If A_4 has no information about whether a given objects satisfies the relation E or not, then the knowledge of agent A_5 is taken into consideration (the methods $A_4.E^{\pm} \wedge A_5.E^-$ and $A_4.E^{\pm} \wedge A_5.E^+$). ∎

3 The Semantics and Computational Method

3.1 Preliminary Notions

The formal semantics of CAKE diagrams involves the use of *simultaneous fixpoints* which are defined by allowing many relations as arguments to standard fixpoint operators. This is required because of the many dependencies between the definitions of different relations. The syntax for fixpoint operators is:

$$\mathsf{lfp}\ R_1(x_1), \ldots, R_k(x_k).A(R_1(x_1), \ldots, R_k(x_k))$$

where $A(R_1(x_1), \ldots, R_k(x_k))$ is stratified[11] w.r.t. all relations $R_1(x_1), \ldots, R_k(x_k)$. Each R_i is called the i-th coordinate of the fixpoint and also denoted by A/i.

Fixpoint formulas have a very nice computational characterization, which allows one to compute simultaneous fixpoints over

[8] More precisely, the standard voting mechanism of CAKE is the following. Suppose that a module is asked a query. (1) If at least one agent contained in a module answers True to the query and none of the agents answers False, the final answer to the query is True. (2) If at least one agent contained in a module answers False to the query and none of the agents answers True, the final answer to the query is False. (3) Otherwise, the answer to the query is Unknown.

[9] That is, the answer determined by this agent is the answer returned by the module.

[10] The symbol $A_4.E^{\pm}$ is an abbreviation for $\neg A_4.E^+ \wedge \neg A_4.E^-$.

[11] The definition of stratification is more or less standard here. The meaning is that recursion does not go through an odd number of negations. For a definition of stratification see e.g. [1].

databases in time polynomial in the size of the database. Namely, given an extensional database B, we have the following definition of the least fixpoint,

$$\langle R_1(x_1), \ldots, R_k(x_k)\rangle = \bigvee_{i\in\omega} A^i(\mathsf{False}, \ldots, \mathsf{False})$$

where brackets $\langle , \rangle$ are used to denote a vector of relations, ω stands for the set of natural numbers, $A^0(\mathsf{False}, \ldots, \mathsf{False}) = \underbrace{\langle \mathsf{False}, \ldots, \mathsf{False}\rangle}_{k-\text{times}}$ and

$A^{i+1}(\mathsf{False}, \ldots, \mathsf{False}) = \langle A^i/1(\mathsf{False}, \ldots, \mathsf{False}), \ldots, A^i/k(\mathsf{False}, \ldots, \mathsf{False})\rangle$.

We will also use the following convention. Assume we have a formula in negation normal form, i.e. negations are only before relation symbols[12]. Then any positive occurrence of any relation symbol, say R, refers to R^+ and any negative occurrence $\neg R$ of R refers to R^-. We also use the convention where all references to boundary regions of relations are eliminated from formulas. The following table describes the equivalences used to do this.

Occurrence in a diagram	Actual Meaning
$R^{\oplus}$	$\neg R^-$
$\neg R^{\oplus}$	R^-
$R^{\ominus}$	$\neg R^+$
$\neg R^{\ominus}$	R^+
$R^{\pm}$	$\neg R^+ \wedge \neg R^-$
$\neg R^{\pm}$	$R^+ \vee R^-$

(1)

3.2 Associating Rules with Diagrams

In this section, we always assume that the rules associated with diagrams are universally quantified over free variables.

3.2.1 Associating Rules with Granular Agents and Adjudicating Agents

Assume we are given a granular or an adjudicating agent diagram labeled with $N : R(\bar{x})$ and containing $A_1(\bar{z_1}), \ldots, A_k(\bar{z_k})$ in its positive part and $B_1(\bar{y_1}), \ldots, B_l(\bar{y_l})$ in its negative part. Then we associate the following rules with the diagram:

$$\begin{aligned}&[\exists\bar{u}.(N.A_1(\bar{z_1}) \vee \ldots \vee N.A_k(\bar{z_k}))] \to N.R^+(\bar{x}) \\ &[\exists\bar{v}.(N.B_1(\bar{y_1}) \vee \ldots \vee N.B_l(\bar{y_l}))] \to N.R^-(\bar{x}),\end{aligned} \tag{2}$$

where:

- $\bar{x} \subseteq (\bar{y}_1 \cup \ldots \cup \bar{y}_l)$ and $\bar{x} \subseteq (\bar{z}_1 \cup \ldots \cup \bar{z}_k)$
- $\bar{v} = [(\bar{y}_1 \cup \ldots \cup \bar{y}_l) - \bar{x}]$ and $\bar{u} = [(\bar{z}_1 \cup \ldots \cup \bar{z}_k) - \bar{x}]$.

Now the definition of the boundary region of the relation defined by N is the following

$$N.R^{\pm}(\bar{x}) \equiv (\neg N.R^+(\bar{x}) \wedge \neg N.R^-(\bar{x})).$$

3.2.2 Associating Rules with Knowledge Modules

Assume we are given a knowledge module diagram labeled by

$$M : R_1(\bar{x}_1), \ldots, R_k(\bar{x}_k).$$

For $1 \le i \le k$, let M_i be the set of all subcomponents of M responsible for delivering the relation R_i and assume M_i does not contain an adjudicating agent for R_i. Then we associate the following set of rules with the diagram, for any $1 \le i \le k$:

$$\begin{aligned}&[\bigvee_{N\in M_i} N.R_i{}^-(\bar{x}_i) \wedge \neg \bigvee_{N\in M_i} N.R_i{}^+(\bar{x}_i)] \to M.R_i{}^-(\bar{x}) \\ &[\bigvee_{N\in M_i} N.R_i{}^+(\bar{x}_i) \wedge \neg \bigvee_{N\in M_i} N.R_i{}^-(\bar{x}_i)] \to M.R_i{}^+(\bar{x}).\end{aligned} \tag{3}$$

In the case where M contains an adjudicating agent A responsible for delivering the relation R_i, we then attach the following rules to the diagram instead of rules (3):

$$\begin{aligned}&A.R_i^+(\bar{x}_i) \to M.R_i^+(\bar{x}_i) \\ &A.R_i^-(\bar{x}_i) \to M.R_i^-(\bar{x}_i).\end{aligned} \tag{4}$$

The definition of the boundary region of the relation defined by M is obtained in the manner used with agent diagrams, i.e. it is given by the following equivalence:

$$M.R^{\pm}(\bar{x}) \equiv (\neg M.R^+(\bar{x}) \wedge \neg M.R^-(\bar{x})).$$

3.3 Obtaining Explicit Definitions of Relations

In order to provide a semantics for relations, we require that the diagrams used are stratified[13]. Let $S, \ldots, T$ be all relations appearing in the heads of rules attached to granular agents, adjudicating agents and knowledge modules, and let B be the conjunction of the rules. Then the following simultaneous fixpoint formula defines relations $S, \ldots, T$:

$$\mathsf{lfp}\ S, \ldots, T.B. \tag{5}$$

The boundary regions of the relations are then obtained using the suitable definitions as specified in the previous section.

Observe that one can obtain inconsistent information in the sense that both the positive and negative part of a relation may contain the same tuple. A nice feature of the approach is that inconsistencies can be checked in time polynomial in the size of the underlying database.

3.4 Computing the Relations

Observe that the definitions of relations obtained in section 3.3 are expressed by means of fixpoint formulas. Using standard techniques for calculating fixpoints, one can easily provide a tractable method for computing the relations. One can even use the method sketched in section 3.1. In addition, if the database domain is linearly ordered then any PTIME query can be modeled by agent diagrams, since recursion within the diagrams is allowed. These results follow easily from standard database theory and may be found in [1].

4 Examples

We now show how the computational method for CAKE, described above, works for the examples considered in section 2.

[12] Any formula is easily transformed to this form.

[13] Let P be a set of rules. By the *dependency graph of* P we mean a graph with vertices labelled by predicates of P and containing two types of edges: (1) there is a *positive edge* $\langle Q, R\rangle$ in the graph iff there is a rule in P in which Q appears positively in the rule's body and R appears in the rule's head; (2) there is a *negative edge* $\langle Q, R\rangle$ in the graph iff there is a rule in P in which Q appears negatively in the rule's body and R appears in the rule's head. The set of rules P is *stratified* if no cycle in its dependency graph contains a negative edge.

Example 4.1 The following is a continuation of Example 2.1. The following set of rules is automatically generated and associated with the diagram of the theory in the example.

$$
\begin{aligned}
&A_1.Q^+(n)\\
&A_2.R^+(n)\\
&[A_3.P^-(x) \lor [A_2.R^+(x) \land \neg A_3.P^+(x)]] \to A_4.P^-(x)\\
&A_3.P^+(x) \to A_4.P^+(x)\\
&A_3.P^- \to A_5.P^-(x)\\
&[A_3.P^+(x) \lor [A_1.Q^+(x) \land \neg A_3.P^-(x)]] \to A_5.P^+(x)\\
&[A_4.P^-(x) \lor A_5.P^-(x)] \land \neg[A_4.P^+(x) \lor A_5.P^+(x)] \to\\
&\qquad M.P^-(x)\\
&[A_4.P^+(x) \lor A_5.P^+(x)] \land \neg[A_4.P^-(x) \lor A_5.P^-(x)] \to\\
&\qquad M.P^+(x).
\end{aligned}
$$

The last two rules represent the standard voting mechanism used by module M.

The relations $A_1.Q^+$, $A_2.R^+$, $A_4.P^-$, $A_4.P^+$, $A_5.P^-$, $A_5.P^+$, $M.P^-$ and $M.P^+$, occurring in the heads of the above rules, are defined by the fixpoint formula given by

$$
\mathsf{lfp}\ A_1.Q^+, A_2.R^+, A_4.P^-, A_4.P^+, A_5.P^-, A_5.P^+, M.P^-, M.P^+.\mathcal{R} \tag{6}
$$

where $\mathcal{R}$ denotes the conjunction of the rules.

Applying the fixpoint computation procedure, we can compute the relations characterized by the simultaneous fixpoint formula (6):

$$
\begin{aligned}
&\{\}\\
&\{A_1.Q^+(n), A_2.R^+(n)\}\\
&\{A_1.Q^+(n), A_2.R^+(n), A_4.P^-(n), A_5.P^+(n)\}.
\end{aligned}
$$

Recall that the query of interest was $P(n)$. As we have already observed, agent A_4 answers False and agent A_5 answers True to the query. However, since it is the default module that is responsible for default inferences about P, and since n satisfies neither the $M.P^+$-coordinate of (6) nor the $M.P^-$-coordinate of (6), we will conclude that the answer to the query $P(n)$ is Unknown. ■

Example 4.2 The following is a continuation of Example 2.2. The following set of rules is automatically generated and associated with the diagram of the theory in the example:

$$
\begin{aligned}
&A_1.A^+(j)\\
&A_2.FTS^+(j)\\
&[A_3.E^-(x) \lor [A_2.FTS^+(x) \land \neg A_3.E^+(x)]] \to A_4.E^-(x)\\
&A_3.E^+(x) \to A_4.E^+(x)\\
&A_3.E^-(x) \to A_5.E^-(x)\\
&[A_3.E^+(x) \lor [A_1.A^+(x) \land \neg A_3.E^-(x)]] \to A_5.E^+(x)\\
&[A_4.E^-(x) \lor [\neg A_4.E^+(x) \land \neg A_4.E^-(x) \land A_5.E^-(x)]] \to\\
&\qquad A_6.E^-(x)\\
&[A_4.E^+(x) \lor [\neg A_4.E^+(x) \land \neg A_4.E^-(x) \land A_5.E^+(x)]] \to\\
&\qquad A_6.E^+(x)\\
&A_6.E^+(x) \to M.E^+(x)\\
&A_6.E^-(x) \to M.E^-(x).
\end{aligned}
$$

The relations occurring in the heads of the above rules are specified by the fixpoint formula given by

$$
\mathsf{lfp}\ \overline{X}.\mathcal{R} \tag{7}
$$

where $\overline{X}$ is the tuple $A_1.A^+$, $A_2.FTS^+$, $A_4.E^+$, $A_4.E^-$, $A_5.E^+$, $A_5.E^-$, $A_6.E^+$, $A_6.E^-$, $M.E^+$, $M.E^-$ and $\mathcal{R}$ is the conjunction of all the rules.

Applying the fixpoint computation procedure, we can compute the relations defined by the simultaneous fixpoint formula (7):

$$
\begin{aligned}
&\{\}\\
&\{A_1.A^+(j), A_2.FTS^+(j)\}\\
&\{A_1.A^+(j), A_2.FTS^+(j), A_4.E^-(j), A_5.E^+(j)\}\\
&\{A_1.A^+(j), A_2.FTS^+(j), A_4.E^-(j), A_5.E^+(j), A_6.E^-(j)\}\\
&\{A_1.A^+(j), A_2.FTS^+(j), A_4.E^-(j), A_5.E^+(j),\\
&\qquad A_6.E^-(j), M.E^-(j)\}.
\end{aligned}
$$

Recall that the query under consideration was $\neg E(j)$. Since j satisfies the $M.E^-$-coordinate of (7), we will conclude that the answer to the query is true. ■

5 Conclusions

We presented a technique and associated tool for incrementally building knowledge structures compositionally by viewing the knowledge structures as confederations of granular and other agent types with dependencies between them. The query mechanism uses this structure when doing inference. Syntactically, the knowledge structures can be viewed as rough relational databases and the rule types as combinations of intensional rules with queries, although the structure used by the knowledge engineer is graphical in nature and has a formal semantics. The queries can be quite complex and local minimization policies applied to the rough database can be associated with individual queries, although this functionality was not discussed in the paper. The method allows one to deal with complex knowledge representation formalisms such as default reasoning and (a rich fragment of) circumscription. The CAKE method has a well-defined semantics and has the following important properties:

- The underlying semantics and computation mechanism ensures that any reasoning expressed by the CAKE diagrams is computable in deterministic polynomial time.
- An open world assumption on the knowledge structures (rough database) can be used, but due to the context part of diagrams one can close the *world* locally. This solution makes diagrams attractive from the point of view of applications in robotics and autonomous systems, where the local closed world assumption is a necessary functionality.

ACKNOWLEDGEMENTS

This work has been supported in part by the WITAS UAV project grant under the Wallenberg Foundation, Sweden. The latter two authors have also been supported by KBN grant 8 T11C 00919.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*, Addison-Wesley Pub. Co., 1996.

[2] P. Doherty, J. Kachniarz, and A. Szałas, 'Using contextually closed queries for local closed-world reasoning in rough knowledge databases', in *Rough-Neuro Computing: Techniques for Computing with Words*, eds., L. Polkowski and A. Skowron. Springer-Verlag, (2002).

[3] W. Łukaszewicz, *Non-Monotonic Reasoning - Formalization of Commonsense Reasoning*, Ellis Horwood Series in Artificial Intelligence, Ellis Horwood, 1990.

[4] R. Reiter, 'A logic for default reasoning', *Artificial Intelligence J.*, **13**, 81–132, (1980).