ELSEVIER

Artificial Intelligence 83 (1996) 229–239

Artificial Intelligence

# Off-line reasoning for on-line efficiency: knowledge bases *

Yoram Moses [a,*], Moshe Tennenholtz [b]

[a] *Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 76100 Israel*

[b] *Faculty of Industrial Engineering and Management, Technion, Israel Institute of Technology, Haifa, 32000 Israel*

Received November 1993; revised July 1994

**Abstract**

The complexity of reasoning is a fundamental issue in AI. In many cases, the fact that an intelligent system needs to perform reasoning on-line contributes to the difficulty of this reasoning. This paper considers the case in which an intelligent system computes whether a query is entailed by the system's knowledge base. It investigates how an initial phase of off-line preprocessing and design can improve the on-line complexity considerably. The notion of an *efficient basis* for a query language is presented, and it is shown that off-line preprocessing can be very effective for query languages that have an efficient basis. The usefulness of this notion is illustrated by showing that a fairly expressive language has an efficient basis. A dual notion of an *efficient disjunctive basis* for a knowledge base is introduced, and it is shown that off-line preprocessing is worthwhile for knowledge bases that have an efficient disjunctive basis.

## 1. Introduction

Many activities in the framework of knowledge representation and reasoning are concerned with the following task: an intelligent agent has a given representation of a system (or of a relevant aspect of the world), and a problem that relates to this representation. Its task is to solve this problem relatively efficiently. Typical examples include planning and computing whether a query is entailed by a given knowledge base. The reasoning in both of these cases can be thought of as on-line reasoning: when an

* Part of the material presented in this paper appeared in *Proceedings IJCAI-93* [9].

* Corresponding author. E-mail: yoram@cs.weizmann.ac.il.


*SSDI* 0004-3702(95)00015-1

input is given, our algorithm should reason about the system and about the problem instance, and should find a solution to the problem. In order to handle such problems, researchers often use general schemes of knowledge representation such as first order logic [6], logic programs [4], semantic networks [10], etc. Moreover, the reasoning is then often carried out using general schemes of reasoning such as resolution for theorem proving, Prolog for logic programs, etc.

Consider the following question: can our agent improve its on-line performance in a case where the system it uses (e.g., its knowledge base or representation of the world) is fixed, and we know ahead of time that the agent is to be asked to solve many problems with respect to this system? Intuitively, it is clear that the answer should be positive; given a fixed system there should invariably be special-purpose algorithms for solving problems with respect to this particular system, instead of using general schemes of reasoning. This answer, however, is not very useful to the agent, without our providing the agent with a way in which it can obtain such special-purpose algorithms. Our aim in this paper is to consider the problem of how an initial phase of off-line preprocessing can serve to reduce the complexity of the agent's on-line behavior. We are especially interested in systems where agents might be presented with many potential problems during the on-line activity. In such a case the agent should be allowed to perform rather extensive preprocessing without increasing the amortized cost per solution significantly. Specifically, we shall investigate the following central context. We consider an intelligent system that needs to compute whether particular formulas (queries) are entailed by its knowledge base. We present the notion of an *efficient basis* for a query language, and show that off-line preprocessing can result in greatly improved on-line behavior for query languages that have an efficient basis. In addition, we consider the notion of an *efficient disjunctive basis* for a knowledge base, and show that off-line preprocessing is very effective for knowledge bases that have an efficient disjunctive basis.

Reactive approaches to problems in AI, related especially to planning, have been suggested in a number of works (see [1,2]). Some other works suggested to compile reactive behaviors in advance (see [11]). However, the task of improving on-line behavior does not need to concentrate only on "real" reactive behavior. On-line behavior might refer more generally to the behavior of an agent that faces various problems after the initialization of the system. The main task is to identify areas where off-line processing can be helpful and to suggest a particular type of solution for each such area. In this work, we apply this idea to query evaluation in knowledge bases.

This paper is organized as follows. Section 2 contains a high-level discussion of how off-line reasoning can be used. In Section 3 we discuss the notion of an efficient basis, a property of query languages that makes preprocessing of knowledge bases for these languages extremely effective. Section 4 presents a fairly expressive query language that has an efficient basis, and considers additional examples in which off-line preprocessing is useful in the context of knowledge bases. In Section 5 we present a property of languages for representing knowledge bases that makes preprocessing of knowledge bases worthwhile. A fairly natural language for representing the contents of a knowledge base is shown to have the desired property. Finally, Section 6 considers other possibilities for useful off-line processing, and provides some concluding remarks.

## 2. Off-line versus on-line reasoning

Consider the well-known problem of determining whether queries are entailed by a knowledge base, as discussed for example in [5]. We assume that we have a knowledge base $KB$ expressed in some logical language, and a query language $QL$ in which queries concerning $KB$ are formulated. Given a query $\alpha$, we are interested in whether $KB \models \alpha$. This problem is intractable in the general case. Moreover, even for tractable queries, the verification process might be very inefficient. There are two main approaches that are discussed in the AI literature for overcoming this difficulty:

(1) Replacing problem solving by model checking [3]: discuss knowledge bases that represent specific models, so that a query only needs to be checked against a model, rather than computing whether it is logically entailed by a knowledge base.

(2) Decreasing the expressive power of the knowledge base and of the query language in order to have more tractable queries.

The first approach is in fact the way in which relational databases are treated in theoretical computer science. The second is concerned with finding good tradeoffs between expressiveness and complexity (as is done in the knowledge base case by [5], or in the case of multi-agent activity by [17]).

Another potential way for decreasing complexity is the following. Assume that any specific query can be verified in time $t$ where $t$ might be large but still feasible (e.g., super-linear but polynomial in the size of the knowledge base $KB$). The question is whether we can find a subset $QL' \subset QL$ of a feasible size, verify off-line for each member $\alpha$ of $QL'$ whether $KB \models \alpha$ (all of this might take a considerable amount of time), and use this off-line processing in order to make the on-line behavior more efficient. In the next sections we illustrate how this approach can be useful. We point to a general set of queries that can be handled in this way, and discuss specific examples. Notice that this approach can be treated as a type of multiple query optimization. However, the context of our query optimization (entailment by knowledge bases instead of retrieval from relational databases), and the actual way in which it is performed (off-line preprocessing instead of clever retrieval of a set of queries after their arrival) will be different from classical multiple query optimization (cf. [12]).

The program described above allows efficient processing of many queries made to a fixed knowledge base. A similar approach can be used for treating a dual problem: processing a fixed query made with respect to different knowledge bases. We shall describe a general class of knowledge bases that can be handled in this way. Although this may be somewhat less powerful than our approach to off-line processing of a query language, there are contexts where this does make good sense.

## 3. Query languages with an efficient basis

In this section we concentrate on off-line reasoning in the case where we have a fixed knowledge base, and queries regarding it arrive in a dynamic fashion. We assume that each query formulated in the query language $QL$ can be verified in time $t$ (generally,

$t$ might be a function of the size of the knowledge base and of the size of the current query). For ease of exposition we will assume that the knowledge base $KB$ and every query $\alpha \in QL$ are formulas in the language $\mathcal{L}$ of propositional logic. [1]

Let us denote the set of primitive propositions in this logic by $X = \{x_1, x_2, \ldots, x_n, \ldots\}$. We use $\mathcal{L}_i$ to denote the set of formulas of $\mathcal{L}$ whose primitive propositions are a subset of $\{x_1, x_2, \ldots, x_i\}$. We will associate with every query language $QL$ an infinite sequence $QL_1 \subseteq QL_2 \subseteq \cdots$, where $QL_i = QL \cap \mathcal{L}_i$. We say that a query language $QL'$ is of *polynomial size* if there exists a fixed polynomial $p$, such that $|QL'_i| < p(i)$, where $|QL'_i|$ denotes the number of elements in $QL'_i$.

As usual, we define the *size* of a formula $\alpha$, denoted by $|\alpha|$, to be the number of symbols appearing in the formula. We assume that it takes time $t(|\alpha|)$ to compute any given query $\alpha$. If we expect to encounter a large number of queries to the same knowledge base, it may be very costly to compute each one of them from scratch. In such a case it would be desirable to identify a small set of queries which, once computed, make computing other queries considerably simpler. More specifically, let us refer to the largest index $m$ of a proposition $x_m$ appearing in a query $\alpha$ as the *cardinality* of $\alpha$. Similarly, the cardinality of the knowledge base is its cardinality when viewed as a single formula. In any given case we can make the cardinality of the knowledge base $KB$ coincide with the number of different propositions that appear in $KB$, by appropriate renaming of the propositions. In the sequel, we shall implicitly take the cardinality of a formula (or of the knowledge base) as the complexity parameter when using the terms "polynomial" or "exponential". We remark that very similar results to the ones we will obtain can be obtained if, instead, we used the size of the formula (or knowledge base). Our choice is mainly for ease of exposition. Our goal will be to evaluate polynomially many queries, so that the on-line evaluation of a query using the data obtained in the preliminary stage is significantly more efficient than the evaluation of a query from scratch. As a result, evaluating a super-polynomial number of queries will be made more efficient with the off-line processing than it is without this preprocessing. In order to do so, we need the following definitions.

**Definition 3.1.** A set $B \subseteq QL$ of queries will be called a *basis* for $QL$ if every query in $QL$ is equivalent to a conjunction of elements of $B$. A basis is called an *efficient basis* if its size (as a sublanguage of $QL$) is polynomial.

Given these definitions, we can now show:

**Theorem 3.2.** *Let $QL$ be a query language, and let $QL'$ be an efficient basis for $QL$. Moreover, let $KB$ be a knowledge base of cradinality $m$ (i.e., $KB \in \mathcal{L}_m$). Finally, let $t$ be an upper bound on the time it takes to compute whether $KB \models \alpha'$ for an arbitrary $\alpha' \in QL'_m$. Then there exists an off-line computation of complexity $O(t \cdot poly(m))$, where poly is a fixed polynomial, after which on-line testing whether $KB \models \alpha$ can be performed in time $O(size(\alpha) \cdot \log m)$ for every $\alpha \in QL_m$.*

[1] The ideas presented in this paper can be generalized to other types of representation as well. We briefly mention such a representation in Section 4.

**Proof.** The amount of time required for verifying all the basis queries is $O(t \cdot poly(m))$, since there are only polynomially many such queries, and we have assumed that each of them can be verified in time $t$. After verifying them we can arrange the results of the off-line processing in a binary tree which is sorted according to the names of the queries in the basis and that contains in each node the information of whether or not the appropriate query in the basis is entailed by $KB$. Now, we can evaluate each query by extracting the relevant information about the appropriate basis queries, and taking the conjunction of the corresponding values. This would be satisfactory since

$$KB \models (\alpha \wedge \beta) \quad \text{iff} \quad \text{both } KB \models \alpha \text{ and } KB \models \beta.$$

Extracting the information about each query in the basis can be done in time which is linear in the depth of the tree, which, in turn, can be made logarithmic in the size of the basis. □

We remark that the size of the knowledge base $KB$ in Theorem 3.2 plays a role only in affecting the function $t$. Once the preprocessing is done, the knowledge base can be ignored, and the complexity of computing entailment of a query is linear in the size of the query and in $\log m$.

Notice that, assuming the size of a query is negligible relative to the size of the knowledge base $KB$ and that $t$ is super-linear in the size of a query, this result shows that in the above-mentioned case we are able to get on-line reasoning that is much more efficient than what can be achieved without appropriate off-line computations.

## 4. Efficient on-line reasoning

In the previous section we showed that off-line preprocessing can be very effective for query languages that have an efficient basis. One wonders, however, whether this family contains any natural query languages that can be used in practice. We now present such a query language. Recall that a CNF formula is a conjunction of clauses each of which contains a disjunction of literals. (A literal is a primitive proposition or the negation of one.) A $k$-CNF formula is a CNF formula where each clause contains no more than $k$ literals. We use $k$-CNF to denote the language of all $k$-CNF formulas. It is not hard to show:

**Theorem 4.1.** *For every $k > 0$, the language $k$-CNF has an efficient basis.*

**Proof.** The basis $QL'$ consists, in this case, of all the $k$-clauses. Recall that each $k$-clause is a disjunction of $k$ literals, each of which can be either positive of negative. Let $QL'_m$ consist of all $k$-clauses of cardinality no greater than $m$. (Thus, the only propositions appearing in $QL'_m$ are $x_1, \ldots, x_m$.) The number of formulas in $QL'_m$ is $2^k\binom{m}{k}$. Since $k$ is a constant, this is $O(m^k)$ which is a polynomial in $m$, as desired. □

Recall that every formula of propositional logic is equivalent to a CNF formula. In particular, every formula is equivalent to a $k$-CNF formula for a sufficiently large

$k$. Moreover, formulas that serve as queries to a knowledge base are likely to be expressible as $k$-CNF formulas for a rather small $k$. The language $k$-CNF is thus a fairly expressive query language in general, for which off-line preprocessing is a useful procedure. In addition, any propositional query can be approximated by $k$-CNF in a natural way. Hence, our approach is complementary to approaches for approximations recently discussed in the AI literature [13]. In existing work in AI the element for which preprocessing is performed is a static element, namely a fixed knowledge base. Our work is complementary in the sense that we process the more dynamic element of the system—the query language as a whole. In Sections 5 and 6 we will see how similar ideas can be used when the dynamic element is the knowledge base.

We remark that Theorem 4.1 can be extended somewhat beyond the purely propositional case. In particular, it applies to universal formulas of the predicate calculus of the form $\forall x_1, \ldots, x_n\ \varphi(x_1, \ldots, x_n)$, where $\varphi$ may contain function symbols and relations, but is syntactically of the form of a $k$-CNF formula. (Here we allow as literals not only primitive propositions, but any term of the predicate calculus.) The result and the proof are the same as in the propositional case. The key point remains the existence of a polynomial basis.

We now consider a concrete class of knowledge bases for which the preprocessing stage on a basis for $k$-CNF described above can be performed using feasible resources, and can yield considerable amortized savings in the on-line computations. Consider the case in which the knowledge base $KB$ consists of a formula in disjunctive normal form (DNF). In this case we can show:

**Proposition 4.2.** *Testing whether a $k$-CNF formula $\varphi$ is entailed by a DNF knowledge base $KB$ is polynomial in $|\varphi| \cdot |KB|$.*

**Proof.** First observe that a knowledge base entails a CNF formula if and only if it entails all the conjuncts in the formula. In addition, observe that a DNF knowledge base entails a formula if and only if each one of its disjuncts entails that formula. Combining the above we obtain the desired result. □

Notice that considering very large knowledge bases and the need for close to real-time response during the on-line activity, the above proposition points to the fact that the on-line reasoning in this case might still be rather inefficient. However, Theorem 3.2 guarantees that with appropriate off-line processing (before any query arrives) testing whether a $k$-CNF formula $\varphi$ is entailed by a DNF knowledge base is linear in $|\varphi| \cdot \log(m)$. This is significantly better than what can be achieved without off-line processing.

The results presented so far illustrate the fact that off-line reasoning can greatly improve the on-line performance of useful AI applications involving knowledge bases. However, a designer that decides to use such off-line reasoning must be careful. A possible drawback of such reasoning might appear when we consider knowledge bases that need to be updated frequently. In such cases the contribution of off-line processing depends on the frequency of updates and on the cost of updating the preprocessing step. Suppose that we have two separate knowledge bases $KB_1$ and $KB_2$ that use the same language, that the query language for both of them is $k$-CNF, and that appropriate

off-line reasoning has been performed for each knowledge base separately. If we want to combine these knowledge bases, there is no general way for combining the respective off-line data on which much effort was spent. The designer will have to investigate whether it is worthwhile to compute all the off-line queries again, or whether in the specific case it is possible and worthwhile to combine the off-line results. It should be emphasized that this problem does not arise when the knowledge base is fixed. In some cases, large parts of the knowledge base can often be considered as fixed, and the above-mentioned approach can then be used in a fairly straightforward manner. Nevertheless, it is of interest to show a particular form of systems where this problem can be handled efficiently even when the knowledge base is not fixed. We now show a particular form of knowledge bases for which this is appropriate. In the next section we will discuss the case in which the query is fixed and the knowledge base may vary.

One motivation for discussing knowledge bases that consist of propositions and not of specific models is the need to represent uncertainty (this issue is thoroughly discussed in [5]). Therefore, it is often reasonable to consider knowledge bases in contexts where updates *increase* the degree of uncertainty in the knowledge base. For example, a knowledge base might contain a hypothesis about the relationships between $x_1, \ldots, x_m$, and there might be another knowledge base that represents another alternative for these relationships. Combining these alternatives corresponds to taking a disjunction of propositional formulas. In such cases the appropriate off-line computations can be easily combined. If two scientists worked on different knowledge bases (hypotheses) using off-line computations, and would like to combine their hypotheses (to see what is entailed if it might be the case that only one of the hypotheses is true), then they can combine their off-line computations easily in order to answer the on-line queries efficiently. Formally, this can be formulated as follows:

**Theorem 4.3.** *Let $KB_1$ and $KB_2$ be knowledge bases, let QL be a query language, and let $QL' \subset QL$ be an efficient basis for QL. Finally, suppose that the cardinality of the formula $(KB_1 \vee KB_2)$ is m (thus, $(KB_1 \vee KB_2) \in \mathcal{L}_m$). Then computing the relevant off-line data for QL with respect to $(KB_1 \vee KB_2)$ given the data with respect to $KB_1$ and the data with respect to $KB_2$ can be done in time polynomial in $|QL'_m|$.*

**Proof.** Consider the construction presented in Theorem 3.2. We have such constructions for both $KB_1$ and $KB_2$. The proof follows from the fact that

$$(KB_1 \vee KB_2) \models \alpha \quad \text{iff} \quad KB_1 \models \alpha \text{ and } KB_2 \models \alpha.$$

Given the above, we can verify for each element $q$ of the basis whether or not it is entailed by $KB_1 \vee KB_2$ by extracting and taking the conjunction of the (already computed) results regarding the entailment of $q$ by $KB_1$ and $KB_2$ respectively. □

## 5. Knowledge bases with an efficient basis

Knowledge bases and queries play dual roles in the problem we have been considering. Indeed, Theorem 4.3 makes use of this duality to a limited extent. In this section we

intend to go further and show how almost everything we have said about off-line processing of queries relative to a fixed knowledge base has a precise analogue for off-line processing of (changing) knowledge bases relative to a fixed query.

Recall that we are identifying knowledge bases with (possibly large) formulas in some logical language. As for query languages, we will associate with every language $KBL$ for expressing knowledge bases, an infinite sequence $KBL_1 \subseteq KBL_2 \subseteq \cdots$, where $KBL_i = KBL \cap \mathcal{L}_i$. Again, we say that a language $KBL'$, in which knowledge bases are expressed, is of *polynomial size* if there exists a fixed polynomial $p$, such that $|KBL'_i| < p(i)$.

**Definition 5.1.** A set $DB \subseteq KBL$ of knowledge bases will be called a *disjunctive basis* for $KBL$ if every query in $KBL$ is equivalent to a disjunction of elements of $DB$. A disjunctive basis is called an *efficient disjunctive basis* if its size (as a sublanguage of $KBL$) is polynomial.

One may wonder whether and when the notion of a *disjunctive* basis for a knowledge base may be of interest. Notice that for it to be relevant in a given application, what is required is that the knowledge base can be treated *as if* it is a disjunction of elements, for the purpose of answering queries. There are cases of relevance to AI in which this is reasonable. Consider a knowledge base that is built from a set of examples of an *a priori* unknown concept $C$, for which we are interested in verifying whether a property $\alpha$ holds. This is a frequent scenario in the context of computational learning theory [18]. In order to check whether $C$ entails $\alpha$, we need to check whether $\alpha$ is entailed by each known example of $C$. Formally, this test is equivalent to the test of whether the disjunction of the examples entails $\alpha$. Therefore, the knowledge base (i.e., the set of examples) need not be represented in a disjunctive form, but in order to answer the query it must be interpreted disjunctively. This gives a general and nontrivial incentive for disjunctive representations.

Given the above definitions and motivation, we can now state and prove the following analogue of Theorem 3.2:

**Theorem 5.2.** *Let KBL be a language in which the content of a knowledge base KB can be expressed, and let KBL′ be an efficient basis for KBL. Moreover, let $\alpha$ be a query and assume that the cardinality of $\alpha$ is $m$ (i.e., $\alpha \in \mathcal{L}_m$). Finally, let $t$ be an upper bound on the time it takes to compute whether $KB' \models \alpha$ for an arbitrary $KB' \in KBL'_m$. Then there exists an off-line computation of complexity $O(t \cdot poly(m))$, where poly is a fixed polynomial, after which on-line testing whether $KB \models \alpha$ can be performed in time $O(|KB| \cdot \log m)$ for every $KB \in KBL_m$.*

**Proof.** The proof is analogous to that of Theorem 3.2. The amount of time required for verifying entailment of $\alpha$ by all the knowledge bases in the basis is $O(t \cdot poly(m))$, since there are only polynomially many such knowledge bases, and we have assumed that each such entailment can be verified in time $t$. After verifying these basic entailments, we can arrange the results of the off-line processing in a binary tree which is sorted according to the names of the knowledge bases in the basis and that contains in each node the

information of whether the appropriate knowledge base in the basis entails $\alpha$ or not. Now, we can evaluate $\alpha$ for a given knowledge base by extracting the relevant information about the appropriate knowledge bases in the basis, and taking the conjunction of the corresponding values. This would be satisfactory since

$$(KB_1 \vee KB_2) \models \alpha \quad \text{iff} \quad KB_1 \models \alpha \text{ and } KB_2 \models \alpha.$$

Extracting the information about each knowledge base in the basis can be done in time which is linear in the depth of the tree, which, in turn, can be made logarithmic in the size of the basis. □

In Section 4 we presented a useful query language—$k$-CNF—for which off-line processing of queries relative to a fixed knowledge base was effective. We now consider an analogous language for which off-line processing of knowledge bases is plausible. Recall that a DNF formula is a disjunction of terms each of which contains a conjunction of literals. A $k$-DNF formula is a DNF formula where each term contains no more than $k$ literals. We use $k$-DNF to denote the language of all $k$-DNF formulas. It is not hard to show:

**Theorem 5.3.** *For every $k > 0$, the language $k$-DNF has an efficient disjunctive basis.*

**Proof.** The proof is completely analogous to that of Theorem 4.1. Details are left to the reader. □

Recall that every formula of propositional logic is equivalent to a DNF formula. Moreover, recall that an *example* in computational learning settings (which constitutes a major motivation for this part of the study) can be interpreted as a term, and the set of examples can therefore be interpreted as a DNF formula (for the purpose of answering queries regarding a concept). In many cases the examples contain some unprescribed entries (see [18]) which makes $k$-DNF an appealing type of representation for our purposes. The language $k$-DNF is thus a fairly expressive language for expressing the contents of a knowledge base, for which off-line preprocessing is a useful procedure.

As was the case for Theorem 4.1, Theorem 5.3 can also be extended somewhat beyond the purely propositional case. In particular, it applies to universal formulas of the predicate calculus of the form $\forall x_1, \ldots, x_n \; \varphi(x_1, \ldots, x_n)$, where $\varphi$ may contain function symbols and relations, but is syntactically of the form of a $k$-DNF formula. (Again, we allow here as literals not only primitive propositions, but any term of the predicate calculus.) The result and the proof are the same as in the propositional case. The key point remains the existence of a polynomial disjunctive basis.

As in the case of off-line processes of query languages with respect to a knowledge base, the use of off-line processing of *KB* languages when the queries of interest may occasionally change must be carefully evaluated. We now show a particular form of systems where the related problem (i.e., combining the off-line data for two different queries) can be handled efficiently. These are, for example, systems where we would like to check whether the concept (i.e., the current examples of a concept) entails a

conjunction of queries, for each of which we have already spent much time on obtaining the off-line data.

**Theorem 5.4.** *Let $\alpha$ and $\beta$ be queries, let KBL be a language in which the contents of a knowledge base KB are described, and let $KBL' \subset KBL$ be an efficient disjunctive basis for KBL. Finally, let m be the cardinality of $(\alpha \wedge \beta)$. Then computing the relevant off-line data for KBL with respect to $(\alpha \wedge \beta)$ given the data with respect to $\alpha$ and the data with respect to $\beta$ can be done in time linear in $|KBL'_m|$.*

**Proof.** Consider the construction presented in Theorem 5.2. We have such constructions for both $\alpha$ and $\beta$. The proof follows from the fact that

$$KB \models \alpha \wedge \beta \quad \text{iff} \quad KB \models \alpha \text{ and } KB \models \beta.$$

Given the above, we can verify for each element $\kappa$ of the basis whether or not $\kappa \models \alpha \wedge \beta$ by extracting and taking the conjunction of the (already computed) results regarding the entailment of $\alpha$ and the entailment of $\beta$ by $\kappa$. □

## 6. Conclusions

This paper suggests the use of off-line processing before the initiation of a system in order to improve the on-line behavior of artificial systems. We investigated this approach in the specific framework of entailment of queries by knowledge bases. We believe that the notion of an efficient basis for a query language has the potential of improving actual performance of query evaluation in knowledge bases, and is a new approach to query optimization. In particular, the fact that this approach applies for the case of the expressive query language $k$-CNF suggests that it may be applicable in practice. We believe that the connection made in Section 5 between off-line processing of languages for knowledge bases with respect to fixed queries and issues that arise naturally in learning theory is also worthy of further investigation.

The general idea of using off-line processing for on-line efficiency can find uses in many other problems in the context of AI. One such context is the framework of artificial social systems, which have been suggested as a paradigm for the design of shared multi-agent environments [7, 8]. An artificial social system can be thought of as a legal system for artificial agents. The purpose of this legal system is to provide an environment in which the agents are able to pursue their individual goals in a fairly compatible fashion. These rules should enable the agents to attain a majority of their goals with a limited amount of interference and a minimal need to coordinate their actions explicitly. The design of a social law can be thought of as an instance of off-line preprocessing whose role is to improve the agents' ability to better attain their goals on-line. A bad set of rules may leave the world too complex to plan in, or might make the agents arrive at deadlocks and bottlenecks, and perhaps need extensive negotiations to resolve conflicts. A good social law will help the agents minimize such on-line trouble. Motivations and examples of the application of off-line design of social laws appear in [7, 8, 14–16].

## Acknowledgements

This work was supported in part by a grant from the US–Israel Binational Science Foundation. The first author was supported by a Helen and Milton A. Kimmelman Career Development Chair. The second author was supported in part by an Eshkol fellowship of the Israeli Ministry of Science and Technology, and later by the Air Force Office of Scientific Research. Part of the research was carried out while the second author was in the Department of Applied Mathematics and Computer Science at the Weizmann Institute of Science, and part while he was in the Computer Science Department at Stanford University.

## References

[1] P. Agre, *The Dynamic Structure of Everyday Life* (Cambridge University Press, Cambridge, 1991).
[2] P. Agre and D. Chapman, Pengi: an implementation of a theory of activity, in: *Proceedings AAAI-87*, Seattle, WA (1987) 268–272.
[3] J.Y. Halpern and M.Y. Vardi, Model checking vs. theorem proving: a manifesto, in: *Proceedings Second International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA (1991) 325–334.
[4] R. Kowalski, Predicate logic as a programming language, in: *Proceedings IFIP Conference*, Stockholm (1974) 569–574.
[5] H.J. Levesque, Logic and the complexity of reasoning, Tech. Rept. KRR-TR-89-2, University of Toronto, Toronto, Ont. (1989).
[6] J. McCarthy and P. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: B. Meltzer and D. Michie, eds., *Machine Intelligence* **4** (Edinburgh University Press, Edinburgh, 1969) 463–502.
[7] Y. Moses and M. Tennenholtz, Artificial social systems part I: basic principles, Tech. Rept. CS90-12, Weizmann Institute (1990).
[8] Y. Moses and M. Tennenholtz, Artificial social systems, *Comput. Artif. Intell.* **14** (1995) 533–562.
[9] Y. Moses and M. Tennenholtz, Off-line reasoning for on-line efficiency, in: *Proceedings IJCAI-93*, Chambery (1993) 490–495.
[10] M. Quillian, Word concepts: a theory and simulation of some basic semantic capabilities, *Behav. Sci.* **12** (1967) 410–430.
[11] M.J. Schoppers, Universal plans for reactive robots in unpredictable environments, in: *Proceedings AAAI-87*, Seattle, WA (1987) 1039–1046.
[12] T.K. Sellis, Multiple-query optimization, *ACM Trans. Database Syst.* **13** (1988) 23–52.
[13] B. Selman and H. Kautz, Knowledge complilation using Horn approximations, in: *Proceedings AAAI-91*, Anaheim, CA (1991) 904–909.
[14] Y. Shoham and M. Tennenholtz, On the synthesis of useful social laws for artificial agent societies, in: *Proceedings AAAI-92*, San Jose, CA (1992) 276–281.
[15] Y. Shoham and M. Tennenholtz, On social laws for artificial agent societies: off-line design, *Artif. Intell.* **73** (1995) 231–252.
[16] M. Tennenholtz, On computational social laws for dynamic non-homogeneous social structures, *JETAI* **7** (1995) 379–390.
[17] M. Tennenholtz and Y. Moses, On cooperation in a multi-entity model, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 918–923.
[18] L.G. Valiant, A theory of the learnable, *Commun. ACM* **27** (1984) 1134–1142.