Electronic Transactions on Artificial Intelligence 129
Vol. 1 (1997): *129-132*
http://www.ep.liu.se/ej/etai/1997/006/

Refereed Research Note

Paolo Liberatore:

Compilability of Domain Descriptions in the Language $\mathcal{A}$

Abstract. *In this note we analyze the possibility of reducing the complexity of entailment in $\mathcal{A}$ by a compilation of the domain description. Since a single domain description D must in general be queried many times with respect to many different queries V, it makes sense to reduce it in a form that allows solving the problem of entailment in polynomial time. Using results from the field of language compilation, we prove that such a compilation is impossible, if we impose the result of compilation to be a polynomial data structure.*

Publication and review history. This research note was advertised in ETAI's Electronic Newsletter on Reasoning about Actions and Change on 31.10.1997, and was included in the News Journal for October. Discussion of the note was invited in the ETAI Area of Reasoning about Actions and Change from that date. (No discussion contributions were received). After this period, the note was *accepted* after due refereeing and according to scientific journal standards by the Electronic Transactions on Artificial Intelligence (ETAI) on 29.7.1998.

Additional questions or comments continue to be invited, and will be posted on the note's interaction page at

http://www.ida.liu.se/ext/etai/received/rac/004/aip.html

Archive and Copyright. This research note will be persistently retained at the URL shown at the top of the present page. The copyright for it belongs to the author. The ETAI publishing agreements specify that it is permitted for anyone to download the note from the net, to print out single copies of it, and to use classroom sets of copies for academic purposes. Please refer to the research-note URL for additional information.

The language $\mathcal{A}$ is one of the most popular languages for representing the effect of actions. Since the seminal paper by Gelfond and Lifschitz [3], many other researchers have worked on extending the language to incorporate the indirect effects of actions, the possibility of concurrent actions, etc. Very recently, two papers appeared on the computational complexity of reasoning about actions, [2] and [5]. The latter regards the complexity of deciding whether a domain description in $\mathcal{A}$ is consistent, and what it implies. These problems have been found to be NP and coNP complete, respectively (for the basic definition of the classes NP and coNP we refer to the work of Johnson [4]). Thus, these problem are intractable, that is, no polynomial algorithm exists for solving them. In order to solve intractable problems, a compilation of part of the data may be useful. A general theory of compilation of intractable problems has been defined in [1]. In this paper we apply the results given there to the problems of $\mathcal{A}$.

The problem of language compilation has been deeply analyzed in AI in the last few years. The general pattern is the following. There is an intractable problem. The input of this problem is composed by two parts. The typical situation is that of a knowledge base KB. We extract information from this knowledge base by checking which facts are true, that is, whether $KB \models Q$, for a fact Q. Even for the propositional case, this problem is coNP complete, thus intractable. In many cases, a single knowledge base KB must be queried many times w.r.t. many different Q's. In such cases it makes sense to compile KB into a new data structure H that allows the solution of the problem $KB \models Q$ in polynomial time. What is required is that the data structure H has size polynomial in the size of the initial knowledge base KB. If this compilation is possible, we say that the problem is *compilable*. A graphical representation of compilation is given in Figure 1.

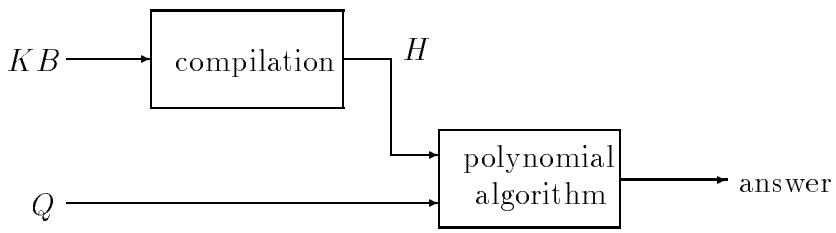

Figure 1: Compilation of a Knowledge Base.

In our case, the knowledge base is a domain description, and the query is a value proposition. However, this basic pattern is present: a domain description may be queried many times w.r.t. to many different value propositions. If it is possible to compile the domain into a data structure H that allows solving the problem of entailment in polynomial time for any possible value proposition, the problem is said to be compilable.

We need some technical definitions and results.

Definition 1 *A problem belongs to* P/poly *if there exists a polynomial p and a family of algorithms $\{A_1, A_2, \ldots\}$, one for each positive integer, such that,*

1. *The algorithm A_i solves the instances of the problem of size i.*
2. *The running time of the algorithm A_i is bounded by $p(i)$.*

The class P/poly is related to the classes of the polynomial hierarchy by the following theorem.

Theorem 1 *If* NP $\subseteq$ P/poly *then* Π_2^p=Σ_2^p=PH.

This theorem says that if the class NP is included in the class P/poly, then the polynomial hierarchy collapses to its second level. This is considered unlikely by complexity researchers.

Using this result we are able to prove the non-compilability of the language $\mathcal{A}$. We use a specific reduction from 3unsat to the problem of entailment. If we are able to find a reduction from the problem of unsatisfiability of a set of clauses Π to the a problem in such a way the fixed part of the problem depends only on the number of variables of Π, then the problem cannot be compiled.

Lemma 1 *Let* Π *be a set of clauses, each composed of three literals.* Π *is unsatisfiable if and only if* $D \models V$, *where*

$$\begin{aligned} D &= \{ \text{initially } \neg F \} \cup \{ A_i \text{ causes } F \text{ if } \neg L_1, \neg L_2, \neg L_3 \mid \\ &\qquad \textit{for each clause } \gamma_i = L_1 \vee L_2 \vee L_3 \in \Pi_X \} \\ V &= F \text{ after } A_1; \ldots; A_m \end{aligned}$$

where Π_X *is the set of all the clauses of three literals over the alphabet* X, *and* $\{A_i\}$ *is a set of actions, one-to-one with the clauses of* Π_X. *The sequence of actions in the value proposition is composed exactly of those* A_i *that are in correspondence with the clauses* γ_i *in* Π.

Proof. Note that D is constructed over Π_X, which depends only on the number of variables in X. The only dependence on the specific set of clauses is in the value proposition V.

Let us now prove the claim. Assume that Π is satisfiable. Let I be a model of Π. Consider the state $\sigma_0 = I$. Since the only value proposition in D is initially $\neg F$, this is a possible initial state, that is, (σ_0, Ψ_D) is a model of D. Note that there is no effect proposition that changes the value of the fluents L_i. Since Π is satisfied, for each clause $\gamma_i = L_1 \vee L_2 \vee L_3$ at least one literal is true. As a result, the action A_i does not change the state when executed, since at least one precondition of the action A_i causes F if $\neg L_1, \neg L_2, \neg L_3$ is false. This holds for each action A_i such that $\gamma_i \in \Pi$, since all the clauses in Π are satisfied by I. As a result, at the end of the sequence $A_1; \ldots; A_m$ the value of F is not modified, since all the clauses corresponding to the A_i in this sequence are in Π.

Let us assume that the set of clauses Π is unsatisfiable. Let (σ_0, Ψ_D) be a model of D. Since Π is unsatisfiable, for each interpretation over $\{L_i\}$ at least one clause must be false in that interpretation. Let $\gamma_i = L_1 \vee L_2 \vee L_3$ be the clause that is falsified by $I = \sigma_0$. Since all the literals it contains are false in I, it follows that all the preconditions of A_i causes F if $\neg L_1, \neg L_2, \neg L_3$ are true, starting from the initial state σ_0. As a result, the fluent F is true at the end of the sequence. This proof does not depend on the specific initial state chosen. As a result, the fluent F is true after the sequence for each possible initial state, thus V is implied by D. □

Lemma 2 *It is possible to give a polynomial reduction from* 3unsat *to entailment in* $\mathcal{A}$ *in such a way the domain description depends only on the size of the formula to be checked if unsatisfiable.*

Proof. The reduction of the previous lemma "almost" satisfies the condition of this lemma. However, the domain description depends on the number of atoms in the considered alphabet, and not on the size of the formula. This problem can easily be overcome by assuming that Π is constructed over an alphabet with a number of elements equal to the size of the formula Π. This is always possible, since there are no constraints imposing that Π must use all the variables of the alphabet. □

Given this lemma, we are able to prove the theorem of non-compilability.

Theorem 2 *It is impossible to compile the domain description D into a polynomial data structure in such a way that entailment is made polynomial, unless the polynomial hierarchy collapses.*

Proof. We proved that there exists two functions f and g such that, given a set of clauses of three literals Π, it holds

$$\Pi \text{ unsatisfiable} \quad \text{iff} \quad f(\|\Pi\|) \models g(\Pi)$$

where $\|\Pi\|$ denotes the size of the formula Π.

Let us assume that the problem of entailment is compilable. We prove that coNP $\subseteq$ P/poly. First we prove that 3unsat is in P/poly. Consider the following family of algorithms:

$$A_i = \text{decide whether } f(i) \models g(\Pi)$$

We proved that this algorithm solves the problem of unsatisfiability of a set of clauses Π of size i. Moreover, since g is polynomial and by hypothesis the problem is compilable, this set of algorithms is bounded by a polynomial. As a result, 3unsat $\in$ P/poly.

Since 3unsat is coNP complete, any other problem in coNP can be reduced to it via a polynomial reduction. As a result, any problem in coNP is also in P/poly. This implies that $\Pi_2^p = \Sigma_2^p = \text{PH}$, which is considered very unlikely in complexity theory. □

References

[1] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. Feasibility and unfeasibility of off-line processing. In *Proc. of ISTCS-96*, pages 100–109. IEEE Computer Society Press, 1996.

[2] T. Drakengren and M. Bjäreland. Reasoning about actions in polynomial time. In *Proc. of IJCAI-97*, pages 1447–1452, 1997.

[3] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *J. of Logic Programming*, 17:301–322, 1993.

[4] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2. Elsevier, 1990.

[5] P. Liberatore. The complexity of the language $\mathcal{A}$. *Electronic J. of Artificial Intelligence*, 1:13–38, 1997. Available at http://www.ep.liu.se/ej/etai/1997/.