*Supported by the
National Science Foundation
Engineering Research Center
Program (NSFD CD 8803012),
Industry and the University*

Complexity, Decidability and Undecidability Results for Rule-Based Expert Systems

by S. Blanksteen, J. Hendler, and D.S. Nau

TR 92-117

Complexity, Decidability and Undecidability Results for Rule-Based Expert Systems[*]

Scott Blanksteen[†] Jim Hendler[‡] Dana S. Nau[§]

Institute for Advanced Computer Studies
University of Maryland

Abstract

We prove the equivalence of domain-independent planning systems and rule-based expert systems. We use this equivalence to examine how the complexity of deriving conclusions in rule-based expert systems depends on the nature of the rules. We show conditions under which conclusion derivation is decidable and undecidable. For those cases where the problem is decidable, we show how the time complexity varies depending on a wide variety of conditions: whether or not function symbols are allowed; whether or not rules may retract facts; whether or not negative conditions are allowed; whether or not the rules are allowed to take arguments; and whether the rules are given as part of the input to the expert system, or instead are fixed in advance.

[*]This work supported in part by NSF Grant IRI-8907890 to the Institute for Systems Research, and a grant from Westinghouse Corp. to the Institute for Systems Research.

[†]Computer Science Dept. and Institute for Systems Research. (301) 405-7027, sib@cs.umd.edu.

[‡]Computer Science Dept., Institute for Systems Research, and Institute for Advanced Computer Studies. (301) 405-2696, hendler@cs.umd.edu.

[§]Computer Science Dept., Institute for Systems Research, and Institute for Advanced Computer Studies. (301) 405-2684, nau@cs.umd.edu.

Table 1: **Complexity of Verifying Rule-Based Expert Systems.**

Lang. restrictions	How rules are given	Allow retraction of facts?	Allow negated conditions?	Telling if a conclusion can be derived	Telling if there is a derivation of length $\leq k$
none	either way	yes/no	yes/no/ no$^{\alpha}$	semidecidable	semidecidable
no function symbols and finitely many constant symbols	given in the input	yes	yes/no	EXPSPACE-complete	NEXPTIME-complete
		no	yes	NEXPTIME-complete	NEXPTIME-complete
			no	EXPTIME-complete	NEXPTIME-complete
			no$^{\alpha}$	PSPACE-comp.	PSPACE-comp.
	fixed (in advance)	yes	yes/no	PSPACE $^{\beta}$	PSPACE $^{\beta}$
		no	yes	NP $^{\beta}$	NP $^{\beta}$
			no	P	NP $^{\beta}$
			no$^{\alpha}$	NLOGSPACE	NP
all predicates are 0-ary (propositions)	given in the input	yes	yes/no	PSPACE-comp.	PSPACE-comp.
		no	yes	NP-complete	NP-complete
			no	P	NP-complete
			no$^{\alpha}$/no$^{\gamma}$	NLOGSPACE-complete	NP-complete
	fixed	yes/no	yes/no	constant time	constant time

$^{\alpha}$No rule has more than one condition.
$^{\beta}$With PSPACE- or NP-completeness for some sets of rules.
$^{\gamma}$Every rule with > 1 condition is the composition of other rules.

1 Introduction

In the current paper, we examine how the complexity of verifying rule-based expert systems, that is, determining whether or not a given conclusion can be derived, depends on the nature of the rules presented to the expert system. We consider expert systems consisting of a working memory, the *blackboard*, and a collection of OPS5-style rules, consisting of lists of conditions and actions. Our results can be summarized as follows:

- If function symbols are allowed or if the language contains infinitely many constant symbols, then determining whether a specific conclusion can be derived is *undecidable* (more specifically, semidecidable). This is true even if we restrict the rules so that (1) they may not retract facts and (2) the condition list of each rule contains at most one (non-negated) atomic formula.

- If no function symbols are allowed and only finitely many constant symbols are allowed, then conclusion derivation is *decidable*, regardless of whether or not rules are allowed to retract facts. In this case, the computational complexity varies from constant time to EXPSPACE-complete, depending on whether or not we allow rules to retract facts or to have negative conditions, whether or not we restrict the predicates to be propositional (i.e., 0-ary), and whether we fix the rules in advance or give them as part of the input. The results are summarized in Table 1.

This paper is organized as follows. Section 2 contains the basic definitions. Sections 3 and 4 prove all of the basic equivalences we need. Section 5 contains the decidability and undecidability results, and Section 6 presents the complexity results. Section 7 contains a discussion of the complexity results. Section 8 contains concluding remarks. The proofs of the theorems and lemmas appear in the appendix.

2 Preliminaries

In this section, we give a precise definition of a rule-based expert system domain, as discussed in this paper.

Definition 2.1 We will assume that the language of an expert system is any first-order language $\mathcal{L}_{es}$ generated by a finite number of function, predicate, and constant symbols. An *atom* in $\mathcal{L}_{es}$ is a predicate symbol together with its arguments. The atom is *ground* if it contains no variable symbols. A *literal* is either an atom (in which case we say the literal is *positive*) or the negation of an atom (in which case we say the literal is *negative*).

Definition 2.2 The *state of the working memory (blackboard)* is a set of ground atoms in the language of the expert system.

The working memory is the set of facts which are known to the expert system at a specific point when a rule is not firing. In a logic interpretation, those facts on the blackboard are true, those not on the blackboard are not true. We consider rule firings to be primitive, indivisible operations; that is, there is no state between the start of a rule firing and the end of the rule firing. We also assume for simplicity that, within a given rule, all retractions occur before any assertions occur.

Definition 2.3 A *rule* α is a 5-tuple (Name(α), Vars(α), Pre(α), Asrt(α), Ret(α)), where

1. Name(α) is simply α;
2. Vars(α) is the set of all variables mentioned in Pre(α);
3. Pre(α) is a finite set of literals, called the *condition list* of α;
4. Asrt(α) and Ret(α) are both finite sets of atoms (possibly non-ground). Asrt(α) is called the *assert list* of α, and Ret(α) is called the *retraction list* of α.

Note that both positive and negative literals may appear in Pre(α), but negative literals may not appear in either Asrt(α) or Ret(α). We will assume for simplicity that all of the input to the expert system for a given trial is provided before any rules are fired.

Definition 2.4 A *rule-based expert system domain* is a pair $\mathcal{D}_{\text{es}} = (B_0, \mathcal{R})$, where B_0 is a blackboard containing the inputs to the expert system and $\mathcal{R}$ is a finite set of rules.

Definition 2.5 A *conclusion set* is a set D of formulas of $\mathcal{L}_{es}$, such that each formula $d \in D$ is an existentially closed conjunct of literals (i.e., the variables, if any, are existentially quantified).

Informally, the conclusion set D is the set of all possible outputs that the expert system might provide that the user is willing to consider to be answers to problems in the expert system domain. For example, if we are discussing a diagnostic expert system, D is the set of all diagnoses the system is capable of making.

Definition 2.6 An *expert system problem* is a triple $\mathcal{P}_{\text{es}} = (B_0, \mathcal{R}, D)$

An expert system problem is a set of rules, inputs and a list of possible outputs.

Definition 2.7 Let $\mathcal{D}_{\text{es}} = (B_0, \mathcal{R})$ be an expert system domain, α be an rule in $\mathcal{R}$, and θ be a substitution that assigns ground terms to each $X_i, 1 \leq i \leq n \in \text{Vars}(\alpha)$. Suppose that the following conditions hold:

$$\begin{aligned}
&\{A\theta | A \text{ is an atom in } \text{Pre}(\alpha)\} \subseteq B; \\
&\{C\theta | \neg C \text{ is a negated literal in } \text{Pre}(\alpha)\} \cap B = \emptyset; \\
&B' = (B - (\text{Ret}(\alpha)\theta)) \cup \text{Asrt}(\alpha)\theta.
\end{aligned}$$

Then we say that α is *θ-executable* from blackboard B, *resulting* in blackboard B'. This is denoted symbolically as $B \overset{\alpha,\theta}{\Longrightarrow} B'$.

θ-executability captures the idea of which rules are executable, given the blackboard state. The notion can be extended to handle varying conflict resolution strategies that may be used.

Definition 2.8 Suppose $\mathcal{P}_{\text{es}} = (B_0, \mathcal{R}, D)$ is an expert system problem and $d \in D$ is a conclusion. A *derivation that concludes* d is a sequence $B_0, \ldots, B_n, B_{n+1}$ of blackboards, a sequence $\alpha_0, \ldots, \alpha_n$ of rules, and a sequence $\theta_0, \ldots, \theta_n$ of substitutions such that $B_0 \overset{\alpha_0,\theta_0}{\Longrightarrow} B_1 \overset{\alpha_1,\theta_1}{\Longrightarrow} B_2 \cdots \overset{\alpha_n,\theta_n}{\Longrightarrow} B_{n+1}$ and d is satisfied by B_{n+1}, i.e. there exists a ground instance of d that is true in B_{n+1}. The *length* of the above derivation is n.

A derivation is simply a sequence of rule instantiations leading from the initial state of the blackboard to one satisfying some state in D. Note that each α_n must have been θ-executable at blackboard B_n.

Definition 2.9 Let $\mathcal{D}_{\text{es}} = (B_0, \mathcal{R})$ be an expert system domain, and let $\mathcal{L}_{es}$ be the language of $\mathcal{D}_{\text{es}}$. Let $\alpha \in \mathcal{R}$ be a rule. Then

1. α is *positive* if $\text{Pre}(\alpha)$ is a finite set of *atoms*, i.e., negations are not present in $\text{Pre}(\alpha)$).

2. α is *retraction-free* if $\text{Ret}(\alpha) = \emptyset$.

3. α is *context-free* $|\text{Pre}(\alpha)| \leq 1$, i.e., $\text{Pre}(\alpha)$ contains at most one atom..

4. α is *side-effect-free* if $|\text{Asrt}(\alpha) \cup \text{Ret}(\alpha)| \leq 1$, i.e., α has at most one action.

5. α is *function-free* if α contains no function symbols.

6. α is *propositional* if every predicate P in α is a propositional symbol, i.e., a predicate symbol of arity 0.

If every rule $\alpha \in \mathcal{R}$ is positive (or retraction-free or context-free or side-effect-free) then we call $\mathcal{D}_{\text{es}}$ positive (or retraction-free or context-free or side-effect-free, respectively). If every rule $\alpha \in \mathcal{R}$ is function-free (or propositional) then we call $\mathcal{L}_{es}$ and $\mathcal{D}_{\text{es}}$ function-free (or propositional, respectively).

Note that if $\mathcal{D}_{\text{es}}$ is propositional then in effect it is also function-free, for even if $\mathcal{L}_{es}$ contains function symbols, no rule will ever use them.

In order to verify the correctness of an expert system, we need to show that it can produce all of the conclusions given in the specification, as well as showing that it does so for the proper inputs, and does not produce these conclusions for other outputs. However, it is typical that expert systems lack complete specifications of input-output behavior; if such a specification existed, it would probably be easier to write a procedural program to perform the task. In light of this, we concentrate here on the question of whether or not the expert system can at least produce all of the desired outputs, as this will be a lower bound on the full verification problem.

Definition 2.10 CONCLUSION EXISTENCE is the following problem:

> Given an expert system problem $\mathcal{P}_{\text{es}} = (B_0, \mathcal{R}, D)$ and a specific conclusion $d \in D$, does there exist a derivation in $\mathcal{P}_{\text{es}}$ that concludes d?

Definition 2.11 CONCLUSION DERIVATION is the following problem:

> Given a expert system problem $\mathcal{P}_{\text{es}} = (B_0, \mathcal{R}, D)$, a specific conclusion $d \in D$, and an integer k encoded in binary, tell whether or not there is a diagnosis in $\mathcal{P}_{\text{es}}$ of length k or less that concludes d.

3 Basic Equivalences

In this paper, our methodology in this paper is to transform the planning problems discussed in [1, 2, 6] into rule-based expert system problems. This allows us to derive results about expert system problems which meet the same criteria as the planning problems to which we are comparing them.

3.1 Planning Definitions

The following definitions are from [1, 2].

Definition 3.1 A *state* is a set of ground atoms.

A state tells us which ground atoms are currently true.

Definition 3.2 A *planning operator* α' is a 4-tuple (Name(α'), Pre(α'), Add(α'), Del(α')), where

1. Name(α') is a syntactic expression of the form $\alpha'(X_1, \ldots, X_n)$ where each X_i is a variable symbol of $\mathcal{L}$;
2. Pre(α') is a finite set of literals, called the *precondition list* of α', whose variables are all from the set $\{X_1, \ldots, X_n\}$;
3. Add(α') and Del(α') are both finite sets of atoms (possibly non-ground) whose variables are taken from the set $\{X_1, \ldots, X_n\}$. Add(α') is called the *add list* of α', and Del(α') is called the *delete list* of α'.

Observe that atoms and negated atoms may occur in the precondition list, but negated atoms may not occur in either the add list or the delete list.

Definition 3.3 A *first-order planning domain* (or simply a *planning domain*) is a pair $\mathcal{D}_{\mathrm{P}} = (S_0, \mathcal{O})$, where S_0 is a state called the *initial state*, and $\mathcal{O}$ is a finite set of planning operators.

Definition 3.4 A *goal* is an existentially closed conjunction of atoms.

Definition 3.5 A *planning problem* is a triple $\mathcal{P}_{\mathrm{P}} = (S_0, \mathcal{O}, G)$, where $(S_0, \mathcal{O})$ is a planning domain and G is a goal.

3.2 Translation Definitions

Definition 3.6 Let $\alpha \in \mathcal{R}$ be any expert system rule, and let $\mathrm{Name}(\alpha) = \alpha$ and $\mathrm{Vars}(\alpha) = (X_1, \ldots, X_n)$. Then the *rule-to-operator* translation function $\mathrm{F}_{\mathrm{ro}} : \alpha \in \mathcal{R} \rightarrow \alpha' \in \mathcal{O}$ is:

- $\mathrm{Name}(\alpha') = \alpha(X_1, \ldots, X_n)$
- $\mathrm{Pre}(\alpha') = \mathrm{Pre}(\alpha)$
- $\mathrm{Add}(\alpha') = \mathrm{Asrt}(\alpha)$
- $\mathrm{Del}(\alpha') = \mathrm{Ret}(\alpha)$

Note that for each rule in $\mathcal{R}$, there is exactly one corresponding operator in $\mathcal{O}$.

Definition 3.7 The *planning domain translation* of an expert system domain $\mathcal{D}_{\mathrm{es}} = (B_0, \mathcal{R})$, denoted $\mathbf{P}(\mathcal{D}_{\mathrm{es}})$, is $(S_0, \mathcal{O})$, where $S_0 = B_0$ and $\mathcal{O}$ is the set of formulae

$$\mathcal{O} = \bigcup_{\alpha \in \mathcal{R}} \mathrm{F}_{\mathrm{ro}}(\alpha).$$

Definition 3.8 The *planning problem translation* of an expert system problem $\mathcal{P}_{\mathrm{es}} = (B_0, \mathcal{R}, D)$, denoted $\mathbf{P}(\mathcal{P}_{\mathrm{es}})$, is $(S_0, \mathcal{O}, G)$, where S_0 and $\mathcal{O}$ are derived from $\mathbf{P}(B_0, \mathcal{R})$ and $G = D$. $\mathcal{L}_p = \mathcal{L}_{es}$.

As we shall show later in Theorem 4.2, the previous two definitions allow us to translate any expert system domain or problem into an equivalent planning domain or problem; the next two definitions, along with Theorem 4.1 allow us to do the converse.

Definition 3.9 Let $\alpha \in \mathcal{O}$ be any planning operator, and let $\mathrm{Name}(\alpha) = \alpha(X_1, \ldots, X_n)$. Then the *operator-to-rule* translation function $\mathrm{F}_{\mathrm{or}} : \alpha \in \mathcal{O} \rightarrow \alpha' \in \mathcal{R}$ is:

- $\mathrm{Name}(\alpha') = \alpha$
- $\mathrm{Vars}(\alpha') = (X_1, \ldots, X_n)$
- $\mathrm{Pre}(\alpha') = \mathrm{Pre}(\alpha)$

- $\mathrm{Asrt}(\alpha') = \mathrm{Add}(\alpha)$
- $\mathrm{Ret}(\alpha') = \mathrm{Del}(\alpha)$

Note that for each operator in $\mathcal{O}$, there is exactly one corresponding rule in $\mathcal{R}$.

Definition 3.10 The *expert system domain translation* of a planning domain $\mathcal{D}_\mathrm{p} = (S_0, \mathcal{O})$, denoted $\mathbf{ES}(\mathcal{D}_\mathrm{p})$, is $(B_0, \mathcal{R})$, where $B_0 = S_0$ and $\mathcal{R}$ is the set of formulae

$$\mathcal{R} = \bigcup_{\alpha \in \mathcal{O}} \mathrm{F}_{\mathrm{or}}(\alpha).$$

Definition 3.11 The *expert system problem translation* of a planning problem $\mathcal{P}_\mathrm{p} = (S_0, \mathcal{O}, G)$, denoted $\mathbf{ES}(\mathcal{P}_\mathrm{p})$, is $(B_0, \mathcal{R}, D)$, where B_0 and $\mathcal{R}$ are derived from $\mathbf{ES}(S_0, \mathcal{O})$ and $D = G$. $\mathcal{L}_{es} = \mathcal{L}_p$.

4 Equivalence Results

In this section, we show that planning is essentially the same as expert system conclusion derivation. We establish this by transforming a planning problem into a expert system domain such that for all goals $g \in G$, the goal g can be achieved by the planner iff there is a derivation for the conclusion g in the expert system.

Theorem 4.1 (Equivalence Theorem 1) Suppose $\mathcal{D}_\mathrm{p} = (S_0, \mathcal{O})$ is a planning domain and $g \in G$ is a goal. Then there is a plan to achieve g from $\mathcal{D}_\mathrm{p}$ iff there exists a derivation of g in $\mathbf{ES}(\mathcal{D}_\mathrm{p})$.

Theorem 4.2 (Equivalence Theorem 2) Suppose $\mathcal{D}_\mathrm{es} = (B_0, \mathcal{R})$ is an expert system domain and $d \in D$ is a goal. Then there exists a derivation to conclude d from $\mathcal{D}_\mathrm{es}$ iff there exists a plan to achieve d in $\mathbf{P}(\mathcal{D}_\mathrm{es})$.

See the appendix for the proofs of the above theorems.

Theorems 4.3 through 4.6 below show that the translation discussed in Theorems 4.1 and 4.2 preserve a number of important properties. The proofs follow directly from the definitions in Section 3.2.

Theorem 4.3 (Preservation of Positivity) If $\mathcal{D}_\mathrm{p}$ is a planning domain, and $\mathbf{ES}(\mathcal{D}_\mathrm{p})$ is its expert system domain translation, then $\mathcal{D}_\mathrm{p}$ is positive iff $\mathbf{ES}(\mathcal{D}_\mathrm{p})$ is positive.

Theorem 4.4 (Preservation of Retraction-freeness) If $\mathcal{D}_\mathrm{p}$ is a planning domain, and $\mathbf{ES}(\mathcal{D}_\mathrm{p})$ is its expert system domain translation, then $\mathcal{D}_\mathrm{p}$ is retraction-free iff $\mathbf{ES}(\mathcal{D}_\mathrm{p})$ is retraction-free.

Theorem 4.5 (Preservation of Function-freeness) If $\mathcal{D}_\mathrm{p}$ is a planning domain, and $\mathbf{ES}(\mathcal{D}_\mathrm{p})$ is its expert system domain translation, then $\mathcal{D}_\mathrm{p}$ is function-free iff $\mathbf{ES}(\mathcal{D}_\mathrm{p})$ is function-free.

Theorem 4.6 (Preservation of Propositionality) If $\mathcal{D}_\mathrm{p}$ is a planning domain, and $\mathbf{ES}(\mathcal{D}_\mathrm{p})$ is its expert system domain translation, then $\mathcal{D}_\mathrm{p}$ is propositional iff $\mathbf{ES}(\mathcal{D}_\mathrm{p})$ is propositional.

5 Decidability and Undecidability

[3] presents a number of decidability and undecidability results for planning systems. Using the equivalences between planning domains and expert system domains that we established in Section 4, we now transport these results over to expert system domains.

Theorem 5.1 (Semi-Decidability Results)

1. $\{d \mid d$ is a conclusion such that there is a deduction to conclude d from $\mathcal{D}_\mathrm{es} = (B_0, \mathcal{R})\}$ is a recursively enumerable subset of the set of all conclusions.

2. given any recursively enumerable collection X of ground atoms (which, of course, are conclusions), there is a positive deletion-free expert system domain $\mathcal{D}_\mathrm{es} = (B_0, \mathcal{R})$ such that $\{A \mid A$ is a ground atom such that there is a derivation to conclude A from $\mathcal{D}_\mathrm{es}\} = X$

Theorem 5.2 The problem "given a positive deletion-free expert system domain $\mathcal{D}_\mathrm{es} = (B_0, \mathcal{R})$, is the set of conclusions diagnosible from $\mathcal{D}_\mathrm{es}$ decidable?" is Π_2^0-complete.

Theorem 5.3 If we restrict $\mathcal{D}_\mathrm{es}$ to be positive, deletion-free, and context-free, then CONCLUSION EXISTENCE is still strictly semi-decidable.

Theorem 5.4 Suppose $\mathcal{D}_{\text{es}} = (B_0, \mathcal{R})$ is a fixed positive, deletion-free expert system domain. Then the problem: "given a conclusion d, does there exist a derivation to conclude d?" is decidable iff the set of goals achievable from $\mathbf{P}(\mathcal{D}_{\text{es}})$ is decidable.

From these results, it is clear that even under severe restrictions, CONCLUSION EXISTENCE is not decidable. In fact, if we allow function symbols, the problem is undecidable, regardless of any other restrictions discussed here.

6 Complexity Results

[2] presents a large number of complexity results for planning systems. Using the equivalences between planning domains and expert system domains that we established in Section 4, we now transport these results over to expert system domains. These results describe how the complexity of determining whether or not a derivation exists for a given conclusion depends on the following conditions:

- whether or not rules may retract facts;
- whether or not negative conditions are allowed;
- whether or not the predicates are restricted to be propositional (i.e., 0-ary);
- whether the rules are given as part of the input to the expert system, or instead are fixed in advance.

As shown in Section 5, if $\mathcal{L}_{es}$ contains any function symbols, or, equivalently, an infinite number of ground atoms, then CONCLUSION EXISTENCE and CONCLUSION DERIVATION are both semidecidable, so, in this section, we assume that $\mathcal{L}_{cs}$ contains no function symbols and only a finite number of ground atoms.

6.1 The Function-Free Case

6.1.1 Rules given in the Input

Theorem 6.1 If $\mathcal{P}_{\text{es}}$ is restricted to be function-free, retraction-free, positive, and context-free, then CONCLUSION EXISTENCE and CONCLUSION DERIVATION are PSPACE-complete.

Theorem 6.2 If $\mathcal{P}_{\text{es}}$ is restricted to be function-free retraction-free and positive, then CONCLUSION EXISTENCE is EXPTIME-complete and CONCLUSION DERIVATION is NEXPTIME-complete.

Theorem 6.3 If $\mathcal{P}_{\text{es}}$ is restricted to be function-free and retraction-free, then CONCLUSION EXISTENCE and CONCLUSION DERIVATION are NEXPTIME-complete.

If rules may retract facts, then whether or not negated conditions are allowed has no effect on the complexity of CONCLUSION EXISTENCE and CONCLUSION DERIVATION.

Theorem 6.4 If $\mathcal{P}_{\text{es}}$ is restricted to be function-free, then CONCLUSION EXISTENCE is EXPSPACE-complete and CONCLUSION DERIVATION is NEXPTIME-complete.

6.1.2 Rules fixed in advance

Theorem 6.5 If we restrict $\mathcal{P}_{\text{es}}$ to be function-free, retraction-free, positive, and context-free and $\mathcal{R}$ to be a fixed set, then CONCLUSION EXISTENCE is in NLOGSPACE and CONCLUSION DERIVATION is in NP.

Theorem 6.6 If we restrict $\mathcal{P}_{\text{es}}$ to be function-free, retraction-free, and positive, and $\mathcal{R}$ to be a fixed set, then CONCLUSION EXISTENCE is in P and CONCLUSION DERIVATION is in NP (with NP-completeness for some sets of rules).

Theorem 6.7 If we restrict $\mathcal{P}_{\text{es}}$ to be function-free and retraction-free, and $\mathcal{R}$ to be a fixed set, then CONCLUSION EXISTENCE and CONCLUSION DERIVATION are both in NP (with NP-completeness for some sets of rules).

If rules may retract facts, then whether or not negated conditions are allowed has no effect on the complexity of CONCLUSION EXISTENCE and CONCLUSION DERIVATION.

Theorem 6.8 If we restrict $\mathcal{P}_{\text{es}}$ to be function-free and $\mathcal{R}$ to be a fixed set, then CONCLUSION EXISTENCE and CONCLUSION DERIVATION are both in PSPACE, with PSPACE-completeness for some sets of rules.

6.2 The Propositional Case

6.2.1 Rules given in the Input

Theorem 6.9 If we restrict $\mathcal{P}_{\text{es}}$ to be function-free, retraction-free, positive, propositional, and restrict each rule to be either context-free or to be a composition of other rules, then CONCLUSION EXISTENCE is NLOGSPACE-complete and CONCLUSION DERIVATION is NP-complete.

Theorem 6.10 If we restrict $\mathcal{P}_{\text{es}}$ to be function-free, retraction-free, positive, propositional, then CONCLUSION EXISTENCE is in P and CONCLUSION DERIVATION is NP-complete.

Theorem 6.11 If we restrict $\mathcal{P}_{\text{es}}$ to be function-free, retraction-free, and propositional, then CONCLUSION EXISTENCE and CONCLUSION DERIVATION are both NP-complete.

If rules may retract facts, then whether or not negated conditions are allowed has no effect on the complexity of CONCLUSION EXISTENCE and CONCLUSION DERIVATION.

Theorem 6.12 If we restrict $\mathcal{P}_{\text{es}}$ to be function-free and propositional, then CONCLUSION EXISTENCE and CONCLUSION DERIVATION are both PSPACE-complete.

6.2.2 Rules Fixed in Advance

If $\mathcal{P}_{\text{es}}$ is an expert system problem where the rules are fixed, then neither the retraction-freeness of the rule nor the existence of negated conditions has any effect on the complexity of CONCLUSION EXISTENCE and CONCLUSION DERIVATION.

Theorem 6.13 If we restrict $\mathcal{P}_{\text{es}}$ to be function-free and propositional and $\mathcal{R}$ to be a fixed set, then CONCLUSION EXISTENCE and CONCLUSION DERIVATION both take constant time.

7 Discussion of Complexity Results

Here we discuss the results described in section 6, and give intuition for the effects of the various conditions on the complexity results for CONCLUSION EXISTENCE for both propositional and non-propositional rule bases. Similar analysis holds for the results about CONCLUSION DERIVATION.

7.1 Propositional Rules

7.1.1 Rules Given in Input

If the rules must be propositional, but are otherwise unrestricted, then a given rule may have to be applied repeatedly during one derivation. There are a polynomial number of rule instantiations, and each blackboard is polynomial in size, so the complexity of CONCLUSION EXISTENCE is PSPACE-complete.

If the rules must be retraction-free, then the set of facts in the working memory continually grows — every fact that appears in the working memory will remain there. This means that any rule instantiation needs to appear only once in a plan, and since there are only a polynomial number of rule instantiations, the complexity is NP-complete.

If we additionally restrict the rules to be positive, then once a rule becomes θ-executable, it will continue to be θ-executable, since no rule can disable another rule by posting a fact to the blackboard. Therefore, the rule firing order no longer affects the conclusions that can be derived. This means there will only be a polynomial number of meaningfully different derviations, so the complexity is in P.

7.1.2 Rules Fixed in Advance

If the rules are fixed and propositional, then we can preprocess the rule base to determine exactly which conclusions can be derived, and how long it will take to derive them, regardless of whether or not the rules may retract facts or whether or not negated conditions are allowed. Therefore, it takes constant time to check whether or not the rule base can derive any given conclusion.

7.2 Non-Propositional Rules

Here we assume that there are no function symbols and only finitely many constants.

7.2.1 Rules Given in Input

With no restrictions on the rules, other than that they have no function symbols, we may need to fire a given rule numerous times to achieve a given conclusion. This forces us to search through all blackboards, which may be doubly exponential in number. Each blackboard is at most exponential in size, so CONCLUSION EXISTENCE is in EXPSPACE.

If we restrict the rules to be retraction-free, then the set of facts on the blackboard after any rule fires is always a superset of the facts on the blackboard before the rule fired. This means that any rule instantiation needs to be fired at most once, and since there are an exponential number of rule instantiations, the complexity is NEXPTIME.

If the rules are now restricted to having no negated conditions, then once a rule becomes θ-executable, it will continue to be θ-executable. Therefore, the order of the rule firings is no longer significant, and the complexity is reduced to EXPTIME.

If we restrict rules to have at most one condition, then backward chaining becomes efficient, as the number of choices at each step remains constant or decreases, and the complexity drops to PSPACE-complete.

7.2.2 Rules Fixed in Advance

If rules are non-propositional and fixed, we may translate them into a set of equivalent rules that are propositional, and otherwise satisfy the same restrictions. That is, if the original rules were retraction-free (positive), the new rules will also be retraction-free (positive). Thus the complexity of CONCLUSION EXISTENCE in the non-propositional, fixed-rules case is approximately the same as in the propositional case with rules as part of the input.

8 Conclusion

We have performed an exhaustive analysis of the complexity of rule-based expert systems derivation. A primary result is that if function symbols are allowed in the language of the expert system, or if infinitely many constant symbols are defined, than the problem is undecidable. The problem remains extremely complex even under fairly severe restrictions on the rules given to the expert system.

References

[1] K. Erol, D. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. 1991. Submitted for publication.

[2] K. Erol, D. Nau, and V. S. Subrahmanian. On the complexity of domain-independent planning. In *Proc. AAAI-92*, pages 381–386, July 1992.

[3] K. Erol, D. Nau, and V. S. Subrahmanian. When is planning decidable? In *Proc. First Internat. Conf. AI Planning Systems*, pages 222–227, June 1992.

[4] C. L. Forgy. The OPS5 user's manual. Technical Report CMU-CS-81-135, Computer Sci. Dept., Carnegie-Mellon University, 1980.

[5] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[6] Nils Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufmann, 1980.

[7] D. A. Waterman. *A Guide to Expert Systems.* Addison-Wesley, Reading MA, 1986.

Appendix

Lemma A.1 Suppose that $\mathcal{D}_{\text{es}} = (B_0, \mathcal{R})$ is any *positive, deletion-free* expert system domain, and

$$B_0 \overset{\alpha_0,\theta_0}{\Longrightarrow} B_1 \overset{\alpha_1,\theta_1}{\Longrightarrow} B_2 \cdots \overset{\alpha_n,\theta_n}{\Longrightarrow} B_{n+1}$$

is a derivation that derives some conclusion d (we really don't care what d is as far as this lemma is concerned). Then

1. $B_0 \subseteq B_1 \subseteq B_2 \cdots \subseteq B_{n+1}$, and
2. if a rule α is θ-executable in state B_j, then α is θ-executable in state B_k for all $k \geq j$.

Proof of Lemma A.1.

1. Immediate consequence of the fact that $\forall \alpha \in \mathcal{R}\text{Ret}(\alpha) = \emptyset$. Hence, for all $0 \leq i \leq n$,
$$B_{i+1} = B_i \cup \text{Asrt}(\alpha_i)\theta_i.$$
2. Suppose α is θ-executable in state B_j.Then $\text{Pre}(\alpha)\theta \subseteq B_j \subseteq B_k$ is true. As $\text{Pre}(\alpha)$ is negation free, the condition that $\{B\theta \mid \neg B \text{ is a negated atom in } \text{Pre}(\alpha)\} \cap B_k = \emptyset$ is immediately satisfied and hence α is executable in state B_k. This completes the proof.

■

Proof of Theorem 4.1 (Equivalence Theorem 1)

$(\rightarrow)$: Suppose $\mathbf{P}(\mathcal{D}_{\text{es}})$ can achieve G_d. Then there is a plan
$$S_0 \overset{\alpha'_0,\theta_0}{\Longrightarrow} S_1 \overset{\alpha'_1,\theta_1}{\Longrightarrow} S_2 \cdots \overset{\alpha'_n,\theta_n}{\Longrightarrow} S_{n+1}$$
that achieves G_d. Then every literal in G_d must be true in state S_{n+1}. We proceed by induction on n.

Base Case 1 (Goal Immediately Achieved). If G_d is achieved in state S_0, then it must be the case that all literals in G_d are true in state S_0. Since, by Definition 3.7, $S_0 = B_0$, the expert system would immediately conclude d.

Base Case 2 $(n = 0)$. If $n = 0$, then there is a plan
$$S_0 \overset{\alpha'_0,\theta_0}{\Longrightarrow} S_1$$
that achieves the goal. That is, $G_d \subseteq S_0 \bigcup \text{Add}(\alpha'_0)$. Since, by Definition 3.7, $\text{Add}(\alpha'_0) = \text{Asrt}(\alpha)$ for some $\alpha \in \mathcal{R}$, the expert system would conclude d via
$$B_0 \overset{\alpha_0,\theta_0}{\Longrightarrow} B_1$$

Inductive Case. If $\mathbf{P}(\mathcal{D}_{\text{es}})$ can achieve G_d in n steps, then there is a plan

$$S_0 \overset{\alpha'_0,\theta_0}{\Longrightarrow} S_1 \overset{\alpha'_1,\theta_1}{\Longrightarrow} S_2 \cdots \overset{\alpha'_n,\theta_n}{\Longrightarrow} S_{n+1}$$

that achieves G_d. If this is the case, then every literal in G_d is true in state S_{n+1}. Since $\mathcal{D}_{\text{es}}$ is positive and deletion-free, it is either the case that every literal in G_d is true in state S_n, or that $(G_d - S_n) \subseteq \text{Add}(\alpha'_n)$. By the inductive hypothesis, there exists an expert system derivation

$$B_0 \overset{\alpha_0,\theta_0}{\Longrightarrow} B_1 \overset{\alpha_1,\theta_1}{\Longrightarrow} B_2 \cdots \overset{\alpha_{n-1},\theta_{n-1}}{\Longrightarrow} B_n \tag{1}$$

such that everything true in state S_n is true in state B_n. By Lemma A.1, everything true in state B_n is also true in state B_{n+1}. So, in the first case, the expert system would conclude d via (1). In the second case, there exists a rule α_n whose planning translation is α'_n. Since by Definition 3.9, $\text{Asrt}(\alpha_n) = \text{Add}(\alpha'_n)$, there exists an expert system derivation

$$B_0 \overset{\alpha_0,\theta_0}{\Longrightarrow} B_1 \overset{\alpha_1,\theta_1}{\Longrightarrow} B_2 \cdots \overset{\alpha_{n-1},\theta_{n-1}}{\Longrightarrow} B_n \overset{\alpha_n,\theta_n}{\Longrightarrow} B_{n+1}$$

in which $\alpha_0 \ldots \alpha_{n-1}$, $\theta_0 \ldots \theta_{n-1}$, and, therefore, $B_0 \ldots B_n$ are the same as in (1), and all literals true in d are true in blackboard B_{n+1}.

$(\longleftarrow)$: The proof in this direction is the mirror image of the other direction. The details are omitted for brevity. ■

Proof of Theorem 4.2 (Equivalence Theorem 2). The proof is similar to the proof of Theorem 4.1. The details are omitted for brevity. ■

Proof of Theorem 6.1. From Theorems 4.1,4.3,4.4 and 4.5 CONCLUSION EXISTENCE with the restrictions that $\mathcal{P}_{\text{es}}$ be function-free, retraction-free, positive, and context-free reduces to PLAN EXISTENCE with these same restrictions. From Theorems 4.2,4.3,4.4 and 4.5, PLAN EXISTENCE with these restrictions reduces to CONCLUSION EXISTENCE with these restrictions. But from Theorem 10 of [2], PLAN EXISTENCE with these restrictions is PSPACE-complete. Thus, so is CONCLUSION EXISTENCE with these restrictions. The proof for CONCLUSION DERIVATION is similar (by reduction to PLAN LENGTH). ■

The proofs of Theorems 6.2–6.13 are similar.