Expressiveness and tractability in knowledge representation and reasoning[1]

HECTOR J. LEVESQUE[2]
Department of Computer Science, University of Toronto, Toronto, Ont., Canada M5S 1A4

AND

RONALD J. BRACHMAN
AT&T Bell Laboratories, 600 Mountain Avenue, 3C-439, Murray Hill, NJ 07974, U.S.A.

Received November 3, 1986
Revision accepted April 8, 1987

A fundamental computational limit on automated reasoning and its effect on knowledge representation is examined. Basically, the problem is that it can be more difficult to reason correctly with one representational language than with another and, moreover, that this difficulty increases dramatically as the expressive power of the language increases. This leads to a tradeoff between the expressiveness of a representational language and its computational tractability. Here we show that this tradeoff can be seen to underlie the differences among a number of existing representational formalisms, in addition to motivating many of the current research issues in knowledge representation.

Key words: knowledge representation, description subsumption, complexity of reasoning, first-order logic, frames, semantic networks, databases.

Cet article étudie une limitation computationnelle fondamentale du raisonnement automatique et examine ses effets sur la représentation de connaissances. A la base le problème tient en ce qu'il peut être plus difficile de raisonner avec un langage de représentation qu'avec un autre et que cette difficulté augmente considérablement à mesure que croît le pouvoir expressif du langage. Ceci donne lieu à un compromis entre le pouvoir expressif d'un langage de représentation et sa tractibilité computationnelle. Nous montrons que ce compromis peut être vu comme l'une des causes fondamentales de la différence qui existe entre nombre de formalismes de représentation existants et peut motiver plusieurs recherches courantes en représentation de connaissances.

Mots clés : représentation de connaissances, complexité du raisonnement, logique du premier ordre, schémas, réseaux sémantiques, bases de données.

[Traduit par la revue]

Comput. Intell. 3, 78–93 (1987)

1. Introduction

This paper examines from a general point of view a basic computational limit on automated reasoning, and the effect that it has on knowledge representation (KR). The problem is essentially that it can be more difficult to reason correctly with one representational language than with another and, moreover, that this difficulty increases as the expressive power of the language increases. There is a tradeoff between the expressiveness of a representational language and its computational tractability. What we attempt to show is that this tradeoff underlies differences among a number of representational formalisms (such as first-order logic, databases, semantic networks, and frames) and motivates many current research issues in KR (such as the role of analogues, syntactic encodings, and defaults, as well as systems of limited inference and hybrid reasoning).

To deal with such a broad range of representational phenomena we must, of necessity, take a considerably simplified and incomplete view of KR. In particular, we focus on its computational and logical aspects, more or less ignoring its history and relevance in the areas of psychology, linguistics, and philosophy. The area of KR is still very disconnected today and the role of logic remains quite controversial, despite what this paper may suggest. We do believe, however, that the tradeoff discussed here is fundamental. As long as we are dealing with computational systems that reason automatically (without any special intervention or advice) and correctly (once we define what *that* means), we will be able to locate where they stand on the tradeoff: They will either be limited in what knowledge they can represent or unlimited in the reasoning effort they might require.

Our computational focus will not lead us to investigate specific algorithms and data structures for KR and reasoning, however. What we discuss is something much stronger, namely, whether or not algorithms of a certain kind can exist at all. The analysis here is at the *knowledge level* (Newell 1981) where we look at the content of what is represented (in terms of what it says about the world) and not the symbolic structures used to represent that knowledge. Indeed, we examine specific representation schemes in terms of what knowledge they can represent, rather than in terms of how they might actually represent it.

In the next section, we discuss what a KR system is for and what it could mean to reason correctly. Next, we investigate how a KR service might be realized using theorem proving in

[1]This is a revised and substantially augmented version of "A Fundamental Tradeoff in Knowledge Representation and Reasoning," by Hector J. Levesque, which appeared in the Proceedings of the Canadian Society for Computational Studies of Intelligence Conference, London, Ontario, May 1984. It includes portions of two other conference papers: "The Tractability of Subsumption in Frame-Based Description Languages," by Ronald J. Brachman and Hector J. Levesque, which appeared in the Proceedings of the American Association for Artificial Intelligence Conference, Austin, Texas, August 1984; and "What Makes a Knowledge Base Knowledgeable? A View of Databases from the Knowledge Level," by the same authors, which appeared in the Proceedings of the First International Workshop on Expert Database Systems, Kiawah Island, South Carolina, October 1984. Much of this paper appeared as a chapter in *Readings in Knowledge Representation* (Morgan Kaufmann Publishers Inc., 1985), edited by the authors.

[2]Fellow of the Canadian Institute for Advanced Research.

first-order logic and the problem this raises. Following this, we present various representational formalisms and examine the special kinds of reasoning they suggest. We concentrate in particular on frame-based description languages, examining in some detail a simple language and a variant. In the case of this pair of languages, the kind of tradeoff we are talking about is made concrete, with a dramatic result. Finally, we draw some tentative general conclusions from this analysis.

2. The role of knowledge representation

While it is generally agreed that KR plays an important role in (what have come to be called) knowledge-based systems, the exact nature of that role is often hard to define. In some cases, the KR subsystem does no more than manage a collection of data structures, providing, for example, suitable search facilities; in others, the KR subsystem is not really distinguished from the rest of the system at all and does just about everything: make decisions, prove theorems, solve problems, and so on. In this section, we discuss in very general terms the role of a KR subsystem within a knowledge-based system.[3]

2.1. *The knowledge representation hypothesis*

A good place to begin our discussion is with what Brian Smith has called the *knowledge representation hypothesis* (Smith 1982):

> Any mechanically embodied intelligent process will be comprised of structural ingredients that (a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and (b) independent of such external semantical attribution, play a formal but causal and essential role in engendering the behaviour that manifests that knowledge.

This hypothesis seems to underlie much of the research in KR, if not most of the current work in Artificial Intelligence in general. In fact, we might think of *knowledge-based systems* as those that satisfy the hypothesis by design. Also, in some sense, it is only with respect to this hypothesis that KR research can be distinguished from any number of other areas involving symbolic structures such as database management, programming languages, and data structures.

Granting this hypothesis, there are two major properties that structures in a knowledge-based system have to satisfy. First of all, it must be possible to interpret them as *propositions* representing the overall knowledge of the system. Otherwise, the representation would not necessarily be of *knowledge* at all, but of something quite different, like numbers or circuits. Implicit in this constraint is that the structures have to be expressions in a language that has a *truth theory*. We should be able to point to one of them and say what the world would have to be like for it to be true. The structures themselves need not *look* like sentences—there are no syntactic requirements on them at all, other than perhaps finiteness—but we have to be able to understand them that way.

A second requirement of the hypothesis is perhaps more obvious. The symbolic structures within a knowledge-based system must play a *causal role* in the behaviour of that system, as opposed to, say, comments in a programming language. Moreover, the influence they have on the behaviour of the system should agree with our understanding of them as propositions representing knowledge. Not that the system has to be aware in any mysterious way of the interpretation of its structures and their connection to the world;[4] but for us to call it knowledge-based, *we* have to be able to understand its behaviour as if it believed these propositions, just as we understand the behaviour of a numerical program as if it appreciated the connection between bit patterns and abstract numerical quantities.

2.2. *Knowledge bases*

To make the above discussion a bit less abstract, we can consider a very simple task and consider what a system facing this task would have to be like for us to call it knowledge-based. The amount of knowledge the system will be dealing with will, of course, be very small.

Suppose we want a system in PROLOG that is able to print the colours of various items. One way to implement that system would be as follows:

```
printColour(snow) :- !, write("It's white.").
printColour(grass) :- !, write("It's green.").
printColour(sky) :- !, write("It's yellow.").
printColour(X) :- write("Beats me.").
```

A slightly different organization that leads to the same overall behaviour is

```
printColour(X) :-
   colour(X,Y), !, write("It's "), write(Y), write(".").
printColour(X) :- write("Beats me.").

colour(snow,white).
colour(grass,green).
colour(sky,yellow).
```

The second program is characterized by explicit structures representing the (minimal) knowledge[5] the system has about colours and is the kind of system that we are calling knowledge-based. In the first program, the association between the object (we understand as) referring to grass and the one referring to its colour is implicit in the structure of the program. In the second, we have an explicit *knowledge base* (or KB) that we can understand as propositions relating the items to their colours. Moreover, this interpretation is justified in that these structures determine what the system does when asked to print the colour of a particular item.

One thing to notice about the example is that it is not the use of a certain programming language or data-structuring facility that makes a system knowledge-based. The fact that PROLOG happens to be understandable as a subset of first-order logic is largely irrelevant. We could probably read the first program "declaratively" and get sentences representing some kind of knowledge out of it; but these would be very strange ones

[3] We should emphasize that we are concentrating on the kind of knowledge representation system that would be used as a component of a larger AI program. KR research also seems to encompass attempts at general models of cognitive behavior. Although some of our comments are applicable even to such models, we are generally ignoring that part of the field here.

[4] Indeed, part of what philosophers have called the *formality condition* is that computation at some level has to be uninterpreted symbol manipulation.

[5] Notice that typical of how the term "knowledge" is used in AI, there is no requirement of *truth*. A system may be mistaken about the colour of the sky but still be knowledge-based. "Belief" would perhaps be a more appropriate term, although we follow the standard AI usage in this paper.

dealing with writing strings and printing colours, not with the colours of objects.

2.3. The knowledge representation subsystem

In terms of its overall goals, a knowledge-based system is not directly interested in what specific structures might exist in its KB. Rather, it is concerned with what the application domain is like—for example, what the colour of grass is. How that knowledge is represented and made available to the overall system is a secondary concern and one that we take to be the responsibility of the KR subsystem. The role of a KR subsystem, then, is to manage a KB for a knowledge-based system and present a picture of the world based on what it has represented in the KB.[6]

If, for simplicity, we restrict our attention to the yes–no questions about the world that a system might be interested in, what is then involved is being able to determine what the KB says regarding the truth of certain sentences. It is not whether the sentence itself is present in the KB that counts, but whether its truth is *implicit* in the KB. Stated differently, what a knowledge representation system has to be able to determine, given a sentence α, is the answer to the following question:

Assuming the world is such that what is believed is true, is α also true?

We will let the notation KB $\models \alpha$ mean that α is implied (in this sense) by what is in the KB.

One thing to notice about this view of a KR system is that an understanding of the service it provides to a knowledge-based system depends only on the truth theory of the language of representation. Depending on the particular truth theory, determining if KB $\models \alpha$ might require not just simple retrieval capabilities, but also *inference* of some sort. This is not to say that the *only* service to be performed by a KR subsystem is question-answering. If we imagine the overall system existing over a period of time, then we will also want it to be able to augment the KB as it acquires new information about the world.[7] In other words, the responsibility of the KR system is to use appropriate symbolic structures to represent knowledge, and to use appropriate reasoning mechanisms both to answer questions and to assimilate new information, *in accordance with the truth theory of the underlying representation language.*

So our view of KR makes it depend only on the semantics of the representation language, unlike other possible accounts that might have it defined in terms of a set of formal symbol manipulation routines (e.g., a proof theory). This is in keeping with what we have called elsewhere a *functional* view of KR (see Levesque (1984*b*) and Brachman *et al.* (1983)), where the service performed by a KR system is defined separately from the techniques a system might use to realize that service.

3. The logical approach

To make a lot of the above more concrete, it is useful to look at an example of the kinds of knowledge that might be available in a given domain and how it might be represented in a KB. The language that will be used to represent knowledge is that of a standard first-order logic (FOL).[8]

3.1. Using first-order logic

The first and most prevalent type of knowledge to consider representing is what might be called simple *facts* about the world, such as

- Joe is married to Sue.
- Bill has a brother with no children.
- Henry's friends are Bill's cousins.

These might be complicated in any number of ways, for example, by including time parameters and certainty factors.

Simple observations such as these do not exhaust what might be known about the domain, however. We may also have knowledge about the *terminology* used in these observations, such as

- *Ancestor* is the transitive closure of parent.
- Brother is *sibling* restricted to males.
- *Favourite-cousin* is a special type of cousin.

These could be called definitions except for the fact that necessary and sufficient conditions might not always be available (as in the last example above). In this sense, they are much more like standard dictionary entries.

The above two sets of examples concentrate on what might be called *declarative* knowledge about the world. We might also have to deal with *procedural* knowledge that focuses not on the individuals and their interrelationships, but on *advice* for reasoning about these. For example, we might know that

- To find the father of someone, it is better to search for a parent and then check if he is male, than to check each male to see if he is a parent.
- To see if x is an ancestor of y, it is better to search up from y than down from x.

One way to think of this last type of knowledge is not necessarily as advice to a reasoner, but as declarative knowledge that deals implicitly with the combinatorics of the domain as a whole.

This is how the above knowledge might be represented in FOL:

1. The first thing to do is to "translate" the simple facts into sentences of FOL. This would lead to sentences like

$$\forall x\ \mathrm{Friend}(\mathrm{henry}, x) \equiv \mathrm{Cousin}(\mathrm{bill}, x)$$

2. To deal with terminology in FOL, the easiest way is to "extensionalize" it, that is, to pretend that it is a simple observation about the domain. For example, the *brother* statement above would become[9]

$$\forall x \forall y\ \mathrm{Brother}(x,y) \equiv (\mathrm{Sibling}(x,y) \wedge \mathrm{Male}(y))$$

3. Typically, the procedural advice would not be represented explicitly at all in a FOL KB, but would show up in the *form* of (1) and (2) above. Another alternative would be to use

[6] As hinted earlier, this is not the only role that we could imagine for such a subsystem, but this approach is consonant with the majority of work in the field.

[7] It is this management of a KB over time that makes a KR subsystem much more than just the implementation of a static deductive calculus.

[8] The use of FOL per se is not an essential feature of the arguments to follow. Any language that allows us to express what we can in FOL would suffice.

[9] This is a little misleading since it will make the *brother* sentence appear to be no different in kind from the one about Henry's friends, though we surely do not want to say that Henry's friends are *defined* to be Bill's cousins.

extra-logical annotations like the kind used in PROLOG or those described in Moore (1982).

The end result of this process would be a first-order knowledge base: a collection of sentences in FOL representing what was known about the domain. A major advantage of FOL is that given a yes–no question also expressed in this language, we can give a very precise definition of KB $\models \alpha$ (and thus, under what conditions the question should be answered *yes*, *no*, or *unknown*):

KB $\models \alpha$ iff every interpretation satisfying all of the sentences in the KB also satisfies α.[10]

There is, moreover, another property of FOL that makes its use in KR even more simple and direct. If we assume that the KB is a finite set of sentences and let KB stand for their conjunction, it can be shown that

KB $\models \alpha$ iff $\vdash$ (KB $\supset \alpha$)

In other words, the question as to whether or not the truth of α is implicit in the KB reduces to whether or not a certain sentence is a *theorem* of FOL. Thus, the question-answering operation becomes one of *theorem proving* in FOL.

3.2. The problem

The good news in reducing the KR service to theorem proving is that we now have a very clear, very specific notion of what the KR system should do. The bad news is that it is also clear that *this service cannot be provided.* The sad fact of the matter is that deciding whether or not a sentence of FOL is a theorem (i.e., the decision problem) is unsolvable. Moreover, even if we restrict the language practically to the point of triviality by eliminating the quantifiers, the decision problem, though now solvable, does not appear to be solvable in anywhere near reasonable time.[11] It is important to realize that this is not a property of particular algorithms that people have looked at but of the *problem* itself: there *cannot* be an algorithm that does the theorem proving correctly in a reasonable amount of time. This bodes poorly, to say the least, for a service that is supposed to be only a part of a larger knowledge-based system.

One aspect of these intractability results that should be mentioned, however, is that they deal with the *worst case* behaviour f algorithms. In practice, a given theorem-proving algorithm may work quite well. In other words, it might be the case that for a wide range of questions, the program behaves properly, even though it can be shown that there will always be short questions whose answers will not be returned for a very long time, if at all.

How serious is the problem, then? To a large extent this depends on the kind of question you would like to ask of a KR subsystem. The worst case prospect might be perfectly tolerable if you are interested in a mathematical application and the kind of question you ask is an open problem in mathematics. Provided progress is being made, you might be quite willing to stop and redirect the theorem prover after a few months if it seems to be thrashing. Never mind worst case behaviour; this might be the *only* case you are interested in.

[10]The assumption here is that the semantics of FOL specify in the usual way what an interpretation is and under what conditions it will satisfy a sentence.

[11]Technically, the problem is now co-NP-complete, meaning that it is strongly believed to be computationally intractable.

But imagine, on the other hand, a robot that needs to know about its external world (such as whether or not it is raining outside or where its umbrella is) before it can act. If this robot has to call a KR system utility as a subroutine, the worst case prospect is much more serious. Bogging down on a logically difficult but low-level subgoal and being unable to continue without human intervention is clearly an unreasonable form of behaviour for something aspiring to intelligence.

Not that "on the average" the robot might not do alright. The trouble is that nobody seems to be able to characterize what an "average" case might be like.[12] As responsible computer scientists, we should not be providing a general inferential service if all that we can say about it is that by and large it will probably work satisfactorily.[13]

If the KR service is going to be used as a utility and is not available for introspection or control, then it had better be *dependable* both in terms of its correctness and the resources it consumes. Unfortunately, this seems to rule out a service based on full theorem proving (in full first-order logic).

3.3. Two pseudosolutions

There are at least two fairly obvious ways to minimize the intractability problem. The first is to push the computational barrier as far back as possible. Research in automatic theorem proving has concentrated on techniques for avoiding redundancies and speeding up certain operations in theorem provers. Significant progress has been achieved here, allowing open questions in mathematics to be answered (Winker 1982; Wos *et al.* 1984). Along similar lines, VLSI and parallel architectural support stands to improve the performance of theorem provers at least as much as it would any search program.

The second way to make theorem provers more usable is to relax our notion of correctness. A very simple way of doing this is to make a theorem-proving program always return an answer after a certain amount of time.[14] If it has been unable to prove either that a sentence or its negation is implicit in the KB, it could assume that it was independent of the KB and answer *unknown* (or maybe reassess the importance of the question and try again). This form of error (i.e., one introduced by an incomplete theorem prover) is not nearly as serious as returning a *yes* for a *no*, and is obviously preferrable to an answer that never arrives. This is of course especially true if the program uses its resources wisely, in conjunction with the first suggestion above.

[12]This seems to account more than anything for the fact that there are so few average case results regarding decidability.

[13]As we noted earlier, not all KR-related research is aimed at providing an inferential component for a larger, knowledge-based system. Work in a similar spirit sometimes has as its goal the realistic modeling of cognitive agents, imperfections and all. While the central concern of this paper is directed less at such research than at that KR work intent on providing a KR service, we believe that this issue of carefully and precisely characterizing the KR system holds equally well for both. To be informative, cognitive models must be correct and timely, no matter what they are modeling. There is a big difference between a precise and predictable model of (say) sloppy reasoning, and a sloppy model of perfect (or other) reasoning. If the model itself is not well understood or may not even be working properly, it is not going to inform us about anything. Thus, while the system being modeled (e.g., a human) may only "by and large work satisfactorily," the *model* must work reliably, predictably, and completely, or it will not do its job.

[14]The resource limitation here should obviously be a function of how important it might be to answer the question either quickly or correctly.

However, from the point of view of KR, both of these are only pseudosolutions. Clearly, the first alone does not help us guarantee anything about an inferential service. The second, on the other hand, might allow us to guarantee an answer within certain time bounds, but would make it very hard for us to tell how seriously to take that answer. If we think of the KR service as reasoning according to a certain logic, then the logic being followed is immensely complicated (compared to that of FOL) when resource limitations are present. Indeed, the whole notion of the KR system calculating what is implicit in the KB (which was our original goal) would have to be replaced by some other notion that went beyond the truth theory of the representation language to include the inferential power of a particular theorem-proving program. In a nutshell, we can guarantee getting an answer, but not necessarily the one we want.

One final observation about this intractability is that it is *not* a problem that is due to the formalization of knowledge in FOL. If we assume that the goal of our KR service is to calculate what is implicit in the KB, then as long as the truth theory of our representation language is upward-compatible with that of FOL, we will run into the same problem. In ~ticular, using English (or any other natural or artificial language) as our representation language does not avoid the problem as long as we can express in it at least what FOL allows us to express.

4. Expressiveness and tractability

It appears that we have run into a serious difficulty in trying to develop a KR service that calculates what is implicit in a KB and yet does so in a reasonable amount of time. One option we have not yet considered, however, is to *limit* what can be in the KB so that its implications are more manageable computationally. Indeed, as we will demonstrate in this section, much of the research in KR can be construed as trading expressiveness in a representation language for a more tractable form of inference. Moreover, unlike the restricted dialects of FOL typical of those analyzed in the logic and computer science literatures (e.g., in terms of nestings of quantifiers), the languages considered here have at least proven themselves quite useful in practice, however contrived they may appear on the surface.

~'. *Incomplete knowledge*

To see where this tradeoff between expressiveness and tractability originates, we have to look at the use of the expressive power of FOL in KR and how it differs from its use in mathematics.

In the study of mathematical foundations, the main use of FOL is in the formalization of infinite collections of entities. So, for example, we have first-order number and set theories that use quantifiers to range over these classes, and conditionals to state what properties these entities have. This is exactly how Frege intended his formalism to be used.

In KR, on the other hand, the domains being characterized are usually finite. The power of FOL is used not so much to deal with infinities, but to deal with *incomplete knowledge* (Moore 1982; Levesque 1982). Consider the kind of facts[15] that might be represented using FOL:

1. $\neg$Student(john).

This sentence says that John is not a student without saying what he is.

2. Parent(sue,bill) $\vee$ Parent(sue,george).

This sentence says that either Bill or George is a parent of Sue, but does not specify which.

3. $\exists x$ Cousin(bill,x) $\wedge$ Male(x).

This sentence says that Bill has at least one male cousin but does not say who that cousin is.

4. $\forall x$ Friend(george,x) $\supset$ $\exists y$ Child(x,y).

This sentence says that all of George's friends have children without saying who those friends or their children are or even if there are any.

The main feature of these examples is that FOL is not used to capture complex details about the domain, but to avoid having to represent details that may not be known. *The expressive power of FOL determines not so much what can be said, but what can be left unsaid.*

For a system that has to be able to acquire knowledge in a piecemeal fashion, there may be no alternative to using all of FOL. But if we can restrict the kind of the incompleteness that has to be dealt with, we can also avoid having to use the full expressiveness of FOL. This, in turn, might lead to a more manageable inference procedure.

The last pseudosolution to the tractability problem, then, is to restrict the logical form of the KB by controlling the incompleteness of the knowedge represented. This is still a pseudosolution, of course. Indeed, provably, there cannot be a *real* solution to the problem. But this one has the distinct advantage of allowing us to calculate exactly the picture of the world implied by the KB, precisely what a KR service was supposed to do. In what follows, we will show how restricting the logical form of a KB can lead to very specialized, tractable forms of inference.[16]

4.2. *Database form*

The most obvious type of restriction to the form of a KB is what might be called *database form*. The idea is to restrict a KB so that it can only contain the kinds of information that can be represented in a standard database. Consider, for example, a very simple database that talks about university courses. It might contain a relation (or record type or whatever) like

COURSE

ID	NAME	DEPT	ENROLL-MENT	INSTRUCTOR
csc248	Programming Languages	Computer Science	42	S. J. Hurtubise
mat100	History of Mathematics	Mathematics	137	R. Cumberbatch
csc373	Artificial Intelligence	Computer Science	853	T. Slothrop
		. . .		

[15]The use of FOL to capture *terminology* or laws is somewhat different. See Brachman and Levesque (1982) for details.

[16]As we have mentioned, there are other ways of dealing with the tradeoff, but the tactic of limiting the form of the representation seems to account for the vast majority of current practice. Indeed, the only style of representation proven so far to scale up to realistic sizes—database technology—falls under this account.

If we had to characterize in FOL the information that this relation contained, we could use a collection of function-free atomic sentences like[17]

COURSE(csc248) DEPT(csc248,ComputerScience) ENROLLMENT(csc248,42) ...
COURSE(mat100) DEPT(mat100,Mathematics) ...
...

In other words, the tabular database format characterizes exactly the positive instances of the various predicates. But more to the point, since our list of FOL sentences never ends up with ones like

DEPT(mat100,Mathematics) $\vee$ DEPT(mat100,History),

the range of uncertainty that we are dealing with is quite limited.

There is, however, additional information contained in the database not captured in the simple FOL translation. To see this, consider, for instance, how we might try to determine the answer to the question:

How many courses are offered by the Computer Science Department?

The knowledge expressed by the above collection of FOL sentences is insufficient to answer this question; nothing about our set of atomic sentences implies that computer science has at least two courses (since csc373 and csc248 could be names of the same individual), and nothing implies that it has at most two courses (since there could be courses other than those mentioned in the list of sentences). On the other hand, from a database point of view, we could apparently successfully answer our question using our miniature database by phrasing it something like

Count c in COURSE where c.DEPT = ComputerScience;

this yields the definitive answer, "2". The crucial difference here, between failing to answer the question at all and answering it definitively, is that we have actually asked *two different questions*. The formal query addressed to the database must be understood as

How many tuples in the COURSE relation have Computer Science in their DEPT field?

This is a question *not* about the world being modelled at all, but about the *data* itself. In other words, the database retrieval version of the question asks about the structures in the database itself, and not about what these structures represent.[18]

To be able to reinterpret the database query as the intuitive question originally posed about courses and departments (rather than as one about tuples and fields), we must account for additional information taking us beyond the stored data itself. In particular, we need FOL sentences of the form

$c_i \neq c_j$

for distinct constants c_i and c_j, stating that each constant represents a unique individual. In addition, for each predicate, we need a sentence similar in form to

$\forall x[\text{COURSE}(x) \supset x = \text{csc248} \vee \ldots \vee x = \text{mat100}]$

saying that the only instances of the predicate are the ones named explicitly.[19] If we now consider a KB consisting of all of the sentences in FOL we have listed so far, a KR system could, in fact, conclude that there were exactly two computer science courses, just like its database management counterpart. We have included in the imagined KB all of the information, both explicit and implicit, contained in the database.

One important property of a KB in this final form is that it is much easier to use than a general first-order KB. In particular, since the first part of the KB (the atomic sentences) does not use negation, disjunction, or existential quantifications, we know the exact instances of every predicate of interest in the language. There is no incompleteness in our knowledge at all. Because of this, *inference reduces to calculation*. To find out how many courses there are, all we have to do is to count how many appropriate tuples appear in the COURSE relation. We do not, for instance, have to reason by cases or by contradiction, as we would have to in the more general case. For example, if we also knew that either csc148 or csc149 or both were computer science courses but that no computer science course other than csc373 had an odd identification number, we could still determine that there were three courses, but not by simply counting. But a KB in database form does not allow us to express this kind of uncertainty and, because of this expressive limitation, the KR service is much more tractable. Specifically, we can represent what is known about the world using just these sets of tuples, exactly like a standard database system. From this perspective, a database is a knowledge base whose limited form permits a very special form of inference.

This limitation on the logical form of a KB has other interesting features. Essentially, what it amounts to is making sure that there is very close structural correspondence between the (explicit) KB and the domain of interest: For each entity in the domain, there is a unique representational object that stands for it; for each relationship that it participates in, there is a tuple in the KB that corresponds to it. In a very real sense, the KB is an *analogue* of the domain of interest, not so different from other analogues such as maps or physical models. The main advantage of having such an analogue is that it can be used directly to answer questions about the domain. That is, the calculations on the model itself can play the role of more general reasoning techniques much the way arithmetic can re-

[17] This is not the only way to characterize this information. For example, we could treat the field names as function symbols or use ID as an additional relation or function symbol. Also, for the sake of simplicity, we are ignoring here integrity constraints (saying, for example, that each course has a unique enrollment), which may contain quantificational and other logical operations, but typically are only used to verify the consistency of the database, not to infer new facts. None of these decisions affect the conclusions we will draw below.

[18] The hallmark, it would appear, of conventional database management is that its practitioners take their role to be providing users access to the data, rather than using the data to answer questions about the world. The difference between the two points of view is especially evident when the database is very incomplete (Levesque 1984*a*).

[19] This is one form of what has been called the *closed-world assumption* (Reiter 1978*b*).

place reasoning with Peano's axioms. The disadvantage of an analogue, however, should also be clear: Within a certain descriptive language, it does not allow anything to be left unsaid about the domain.[20] In this sense, an analogue representation can be viewed as a special case of a propositional one where the information it contains is relatively complete.

4.3. *Logic-program form*

The second restriction on the form of a KB we will consider is a generalization of the previous one that is found in programs written in PROLOG, PLANNER, many production systems, and related languages. A KB in logic-program form also has an explicit and an implicit part. The explicit KB in a PROLOG program is a collection of first-order sentences (called Horn sentences) of the form

$$\forall x_1 \ldots x_n [P_1 \wedge \ldots \wedge P_m \supset P_{m+1}]$$

where $m \geq 0$ and each P_i is atomic. In the case where $m = 0$ and the arguments to the predicates are all constants, the logic-program form coincides with the database form. Otherwise, because of the possible nesting of functions, the set of relevant terms (whose technical name is the *Herbrand universe*) is much larger and may be infinite.

As in the database case, if we were only interested in the universe of terms, the explicit KB would be sufficient. However, to understand the KB as being about the world, but in a way that is compatible with the answers provided by a PROLOG processor, we again have to include additional facts in an implicit KB. In this case, the implicit KB is normally infinite since it must contain a set of sentences of the form $(s \neq t)$, for any two distinct terms in the Herbrand universe. As in the database case, it must also contain a version of the closed-world assumption which is now a set containing the negation of every ground atomic sentence not implied by the Horn sentences in the explicit KB.

The net result of these restrictions is a KB that once again has complete knowledge of the world (within a given language), but this time, may require inference to answer questions.[21] The reasoning in this case, is the *execution* of the logic program. For example, given an explicit PROLOG KB consisting of

```
parent(bill,mary).
parent(bill,sam).
mother(X,Y) :- parent(X,Y), female(Y).
female(mary).
```

we know exactly who the mother of Bill is, but only after having executed the program.

In one sense, the logic-program form does not provide any computational advantage to a reasoning system since determining what is in the implicit KB is, in general, undecidable.[22] On the other hand, the form is much more manageable than in the general case since the necessary inference can be split very nicely into two components: a *retrieval* component that extracts (atomic) facts from a database by pattern-matching and a *search* component that tries to use the nonatomic Horn sentences to complete the inference. In actual systems like PROLOG and PLANNER, moreover, the search component is partially under user control, giving him the ability to incorporate some of the kinds of procedural knowledge (or combinatoric advice) referred to earlier. The only purely automatic inference is the retrieval component.

This suggests a different way of looking at the inferential service provided by a KR system (without even taking into account the logical form of the KB). Instead of automatically performing the full deduction necessary to answer questions, a KR system could manage a *limited form of inference* and leave to the rest of the knowledge-based system (or to the user) the responsibility of intelligently completing the inference. As suggested in Frisch and Allen (1982), the idea is to take the "muscle" out of the automatic component and leave the difficult part of reasoning as a problem that the overall system can (meta-)reason about and plan to solve (Genesereth 1983; Smith and Genesereth 1985).

While this may be a promising approach, especially for a KB of a fully general logical form, it does have its problems. First of all, it is far from clear what primitives should be available to a program to extend the reasoning performed by the KR subsystem. It is not as if it were a simple matter to generalize the meager PROLOG control facilities to handle a general theorem prover, for example.[23] The search space in this case seems to be much more complex.

Moreover, it is not clear what the KR service itself should be. If all a KR utility does is perform explicit retrieval over sentences in a KB, it would not be much help. For example, if asked about $(p \vee q)$, it would fail if it only had $(q \vee p)$ in the KB. What we really need is an automatic inferential service that lies somewhere between simple retrieval and full logical inference. But finding such a service that can be motivated *semantically* (the way logical deduction is) and defined independently of how any program actually operates is a nontrivial matter, though we have taken some steps towards this in Levesque (1984*c*) (and see Patel-Schneider (1985, 1986)).

4.4. *Semantic-network form*

Semantic networks and similar hierarchic representational frameworks have been in common use in AI for perhaps 20 years. The form of such representations has been apparently dictated by need and a somewhat natural fit to problems under consideration in the field. But, as we shall see next, semantic networks can also be viewed as making a trade of expressive power for a kind of computational tractability.

A first observation about a KB in what we will call "semantic-network form" is that it contains only unary and binary predicates. For example, instead of representing the fact that John's grade in cs100 was 85 by

[20] The same is true for the standard analogues. One of the things a map does not allow you to say, for example, is that a river passes through one of two widely separated towns, without specifying which. Similarly, a plastic model of a ship cannot tell us that the ship it represents does not have two smokestacks, without also telling us how many it does have. This is not to say that there is *no uncertainty* associated with an analogue, but that this uncertainty is due to the coarseness of the analogue (e.g., how carefully the map is drawn) rather than to its content.

[21] Notice that it is impossible to state in a KB of this form that $(p \vee q)$ is true without stating which, or that $\exists x\, P(x)$ is true without saying what that x is. However, see the comments below regarding the use of encodings.

[22] In other words, determining if a ground atomic sentence is implied by a collection of Horn sentences (containing function symbols) is undecidable. This is not true, however, if the Herbrand universe is finite, the case that arises almost exclusively in the type of production system used in expert systems. In fact, in the propositional case, it is not hard to prove that the implicit KB can be calculated in linear time.

[23] Though see Stickel (1984) for some ideas in this direction.

Grade(john, cs100, 85)

we would postulate the existence of objects called "grade-assignments" and represent the fact about John in terms of a particular grade-assignment g-a1 as

$$\text{Grade-assignment(g-a1)} \wedge \text{Student(g-a1,john)} \wedge \text{Course(g-a1,cs100)} \wedge \text{Mark(g-a1,85)}$$

This part of a KB in semantic-network form is also in database form: a collection of function-free ground atoms, sentences stating the uniqueness of constants, and the closed-world assumption.

The main feature of a semantic net (and of the frame form below), however, is not how individuals are handled, but the treatment of the ("generic") predicates (the unary ones we will call *types*, the binary ones we will call *attributes*[24]). First of all, the types are organized into a taxonomy, which, for our purposes, can be represented by a set of sentences of the form[25]

$$\forall x[B(x) \supset A(x)]$$

Thus the basic skeleton of the taxonomy is provided by a universally quantified conditional, or "*is-a*" connection. For example, "Student IS-A Person" would amount to a statement that all Students are Persons, or

$$\forall x[\text{Student}(x) \supset \text{Person}(x)]$$

The second kind of sentence in the generic KB places a constraint on an attribute as it applies to instances of a type:

$$\forall x[B(x) \supset \exists y(R(x,y) \wedge V(y))]$$

or

$$\forall x[B(x) \supset R(x,c)]^{26}$$

This latter form corresponds to "value restriction" in KL-ONE and other languages. For example, equating Graduate with "Person with an Undergraduate Degree" would be the equivalent of

$$\forall x[\text{Graduate}(x) \supset \exists y(\text{Degree}(x,y) \wedge \text{UndergraduateDegree}(y))]$$

This completes the semantic-network form.

One property of a KB in this form is that it can be represented as a labelled directed graph (and displayed in the usual way). The nodes are either constants or types, and the edges are either labelled with an attribute or with the special label *is-a*.[27] The significance of this graphical representation is that it allows certain kinds of inference to be performed by simple graph-searching techniques. For example, to find out if a particular individual has a certain attribute, it is sufficient to search from the constant representing that individual, up *is-a* links, for a node having an edge labelled with the attribute. By placing the attribute as high as possible in the taxonomy, all individuals below it can *inherit* the property. Computationally, any mechanism that speeds up this type of graph-searching can be used to improve the performance of inference in a KB of this form.

In addition, the graph representation suggests different kinds of inference that are based more directly on the structure of the KB than on its logical content. For example, we can ask how two nodes are related and answer by finding a path in the graph between them. Given, for instance, Clyde the elephant and Jimmy Carter, we could end up with an answer saying that Clyde is an elephant and that the favourite food of elephants is peanuts which is also the major product of a farm owned by Jimmy Carter. A typical method of producing this answer would be to perform a "spreading activation" search beginning at the nodes for Clyde and Jimmy. Obviously, this form of question would be very difficult to answer for a KB that was not in semantic-network form.[28]

For better or worse, the appeal of the graphical nature of semantic nets has led to forms of reasoning (such as default reasoning (Reiter 1978*a*)) that do not fall into standard logical categories and are not yet very well understood (Etherington and Reiter 1983).[29] This is a case of a representational notation taking on a life of its own and motivating a completely different style of use not necessarily grounded in a truth theory. It is unfortunately much easier to develop an algorithm that appears to reason over structures of a certain kind than to *justify* its reasoning by explaining what the structures are saying about the world.

This is not to say that defaults are not a crucial part of our knowledge about the world. Indeed, the ability to abandon a troublesome or unsuccessful line of reasoning in favour of a default answer intuitively seems to be a fundamental way of coping with incomplete knowledge in the presence of resource limitations. The problem is to make this intuition precise. Paradoxically, the best formal accounts we have of defaults (such as Reiter (1980)) would claim that reasoning with them is even *more difficult* than reasoning without them, so research remains to be done (but see Lifschitz (1985)).

One final observation concerns the elimination of higher arity predicates in semantic networks. It seems to be fairly commonplace to try to sidestep a certain generality of logical form by introducing special representational objects into the domain. In the example above, a special "grade-assignment" object took the place of a 3-place predicate. Another example is the use of encodings of sentences as a way of providing (what appears to be) a completely extensional version of modal logic (Moore 1980).[30] Not that exactly the same expressiveness is preserved in these cases; but what *is* preserved is still fairly

[24] We use "type" and "attribute" here for consistency. In some systems (like KL-ONE (Brachman and Schmolze 1985)) the former are called "concepts" and the latter "roles."

[25] See Brachman (1983) for a discussion of some of the subtleties involved here.

[26] There are other forms possible for this constraint. For example, we might want to say that *every R* rather than *some R* is a *V*. See also Hayes (1979). For the variant we have here, however, note that the KB is no longer in logic-program form.

[27] Note that the interpretation of an edge depends on whether its source and traget are constants or types. For example, from a constant *c* to a type *B*, *is-a* means *B*(*c*), but from a type *B* to a type *A*, it is a taxonomic sentence (again, see Brachman (1983)).

[28] Quillian (1968) proposed a "semantic intersection" approach to answering questions in his original work on semantic nets. See also Collins and Loftus (1975) for follow-up work on the same topic.

[29] A simple example of a default would be to make *elephant* have the colour *grey* but to allow things below elephant (such as *albino-elephant*) to be linked to a different colour value. Determination of the colour of an individual would involve searching up for a value and stopping when the first one is found, allowing it to preempt any higher ones. See also Brachman (1985) and Touretzky (1986).

[30] Indeed, some modern semantic network formalisms (such as Shapiro (1979)) actually include all of FOL by encoding sentences as terms.

mysterious and deserves serious investigation, especially given its potential impact on the tractability of inference.

4.5. *Frame-description form*

The final form we will consider, the frame-description form, is mainly an elaboration of the semantic-network one. The emphasis, in this case, is on the structure of types themselves (usually called *frames*), particularly in terms of their attributes (called *slots*). Typically, the kind of detail involved with the specification of attributes includes

1. *values*, stating exactly what the attribute of an instance should be. Alternatively, the value may be just a *default*, in which case an individual inherits the value provided he does not override it.
2. *restrictions*, stating what constraints must be satisfied by attribute values. These can be *value* restrictions, specified by a type that attribute values should be instances of, or *number* restrictions, specified in terms of a minimum and a maximum number of attribute values.
3. *attached procedures*, providing procedural advice on how the attribute should be used. An *if-needed* procedure says how calculate attribute values if none have been specified; an *if-added* procedure says what should be done when a new value is discovered.

Like semantic networks, frame languages tend to take liberties with logical form and the developers of these languages have been notoriously lax in characterizing their truth theories (Brachman 1985; Etherington and Reiter 1983; Hayes 1979). What *we* can do, however, is restrict ourselves to a non-controversial subset of a frame language that supports descriptions of the following form:

```
(Student
  with a dept is computer-science and
  with ≥ 3 enrolled-course is a
    (Graduate-Course
      with a dept is a Engineering-Department))
```

This is intended to be a structured type that describes computer science students taking at least three graduate courses in departments within engineering. If this type had a name (say A), we could express the type in FOL by a "meaning postulate" of form

$$\begin{aligned}\forall x\, A(x) \equiv [&\text{Student}(x) \wedge \text{dept}(x, \text{computer-science}) \wedge \\ &\exists y_1 y_2 y_3 (y_1 \neq y_2 \wedge y_1 \neq y_3 \wedge y_2 \neq y_3 \wedge \\ &\quad \text{enrolled-course}(x, y_1) \wedge \text{Graduate-Course}(y_1) \wedge \\ &\quad\quad \exists z(\text{dept}(y_1, z) \wedge \text{Engineering-Department}(z)) \wedge \\ &\quad \text{enrolled-course}(x, y_2) \wedge \text{Graduate-Course}(y_2) \wedge \\ &\quad\quad \exists z(\text{dept}(y_2, z) \wedge \text{Engineering-Department}(z)) \wedge \\ &\quad \text{enrolled-course}(x, y_3) \wedge \text{Graduate-Course}(y_3) \wedge \\ &\quad\quad \exists z(\text{dept}(y_3, z) \wedge \text{Engineering-Department}(z)))]\end{aligned}$$

Similarly, it should be clear how to state equally clumsily[31] in FOL that an individual is an instance of this type.

One interesting property of these structured types is that we do not have to state explicitly when one of them is below another in the taxonomy. The descriptions themselves implicitly define a taxonomy of *subsumption*, where type A subsumes type B if, by virtue of the form of A and B, every instance of B must be an instance of A. For example, without any world knowledge, we can determine that the type *Person* subsumes

(Person **with every** male friend **is a** Doctor)

which in turn subsumes

(Person **with every** friend **is a**
  (Doctor **with** a specialty **is** surgery))

Similarly,

(Person **with** ≥2 children)

subsumes

(Person **with** ≥3 male children).

Also, we might say that two types are *disjoint* if no instance of one can be an instance of the other. An example of disjoint types is

(Person **with** ≥3 young children)

and

(Person **with** ≤2 children).

Analytic relationships like subsumption and disjointness are properties of structured types that are not available in a semantic net where all of the types are atomic.

There are very good reasons to be interested in these analytic relationships (Brachman and Levesque 1982). In KRYPTON (Brachman *et al.* 1983, 1985), a full first-order KB is used to represent facts about the world. However, subsumption and disjointness information is made available without having to enlarge the KB with a collection of meaning postulates representing the structure of the types, but rather via a separate "terminological component" based on a language in frame-description form. This is significant because, while subsumption and disjointness can be defined in terms of logical implication,[32] there are good special-purpose algorithms for calculating these relationships in KRYPTON's frame-description language.[33] Again, because the logical form is sufficiently constrained, the required inference can be much more tractable.

4.6. *A detailed example of the tradeoff*

As it turns out, frame-description languages and the subsumption inference provide a rich domain for studying the tradeoff between expressiveness and tractability. To illustrate this, we will consider in some detail a simple frame-description language, which we will call $\mathcal{FL}$.[34]

$\mathcal{FL}$ has the following grammar:

```
⟨type⟩      ::= ⟨atom⟩
              | (AND ⟨type₁⟩ . . . ⟨typeₙ⟩)
              | (ALL ⟨attribute⟩ ⟨type⟩)
              | (SOME ⟨attribute⟩)
⟨attribute⟩ ::= ⟨atom⟩
              | (RESTRICT ⟨attribute⟩ ⟨type⟩)
```

[31] What makes these sentences especially awkward in FOL is the number restrictions. For example, the sentence *"There are a hundred billion stars in the Milky Way Galaxy"* would be translated into an FOL sentence with about 10^{22} conjuncts.

[32] Specifically, type A subsumes type B iff the meaning postulates for A and B logically imply the sentence, $\forall x[B(x) \supset A(x)]$.

[33] In particular, see the next section. Also, see Stickel (1985) for details on speedups achieved in this fashion.

[34] Please note that the style of this section constitutes a significant departure from that of previous sections, presenting enough technical detail to make the point truly concrete.

Intuitively, we think of types in $\mathcal{FL}$ as representing (sets of) individuals, and attributes as representing relations between individuals.

While the linear syntax is a bit unorthodox, $\mathcal{FL}$ is actually a distillation of the operators in typical frame languages; in particular, it is the frame-description kernel derived from years of experience with languages like KL-ONE (Brachman and Schmolze 1985) and KRYPTON (Brachman and Levesque 1982; Brachman *et al.* 1983, 1985):

- Atoms are the names of primitive (undefined) types.
- AND constructions represent conjoined types, so, for example, (AND adult male person) would represent the concept of something that was at the same time an adult, a male, and a person (i.e., a man). In general, x is an (AND $t_1 t_2 \ldots t_n$) iff x is a t_1 and a t_2 and ... and a t_n. This allows us to put several properties (i.e., supertypes or attribute restrictions) together in the definition of a type.
- The ALL construct provides a type-restriction on the values of an attribute (x is an (ALL a t) iff each a of x is a t). Thus (ALL child doctor) corresponds to the concept of something all of whose children are doctors. It is a way to *restrict* the value of a slot at a frame (a "value restriction" in KL-ONE).
- The SOME operator guarantees that there will be at least one value for the attribute named (x is a (SOME a) iff x has at least one a). For instance, (AND person (SOME child)) would represent the concept of a parent. This is a way to *introduce* a slot at a frame.

Note that in the more common frame languages, the ALL and SOME are not broken out as separate operators, but instead, either every attribute restriction is considered to have *both* universal and existential import, or exclusively one or the other (or it may even be left unspecified).[35] Our language allows for arbitrary numbers of attribute values, and allows the SOME and ALL restrictions to be specified independently.

- Finally, the RESTRICT construct accounts for attributes constrained by the types of their values, e.g., (RESTRICT child male) for a child who is a male, that is, a son (in general, y is a (RESTRICT a t) of x iff y is an a of x and y is a t).

The $\mathcal{FL}$ language can be considered a simplified (though less readable) version of the frame-based language used in the previous section. So, for example, where we would previously have written a description like

(person **with every** male friend **is a**
  (doctor **with a** specialty))

the equivalent $\mathcal{FL}$ type is written as

(AND person (ALL (RESTRICT friend male)
  (AND doctor (SOME specialty))))

To specify exactly what these constructs mean, we now briefly define a straightforward extensional semantics for $\mathcal{FL}$. As a result, we will provide a precise definition of subsumption. This will be done as follows: imagine that associated with each description is the set of individuals (individuals for types, pairs of individuals for attributes) that it describes. Call that set the *extension* of the description. Notice that by virtue of the structure of descriptions, their extensions are not independent (for example, the extension of (AND t_1 t_2) should be the intersection of those of t_1 and t_2). In general, the structures of two descriptions can imply that the extension of one is always a superset of the extension of the other. In that case, we will say that the first *subsumes* the second (so, in the case just mentioned, t_1 would be said to subsume (AND t_1 t_2)).

More formally, let $\mathcal{D}$ be any set and $\mathcal{E}$ be any function from types to subsets of $\mathcal{D}$ and attributes to subsets of the Cartesian product, $\mathcal{D} \times \mathcal{D}$. So

$\mathcal{E}[t] \subseteq \mathcal{D}$ for any type t

and

$\mathcal{E}[a] \subseteq \mathcal{D} \times \mathcal{D}$ for any attribute a

We will say that $\mathcal{E}$ is an *extension function* over $\mathcal{D}$ if and only if

1. $\mathcal{E}[(\text{AND } t_1 \ldots t_n)] = \cap_i \mathcal{E}[t_i]$
2. $\mathcal{E}[(\text{ALL } a\ t)] = \{x \in \mathcal{D} \mid \forall y \text{ if } \langle x,y\rangle \in \mathcal{E}[a] \text{ then } y \in \mathcal{E}[t]\}$
3. $\mathcal{E}[(\text{SOME } a)] = \{x \in \mathcal{D} \mid \exists y[\langle x,y\rangle \in \mathcal{E}[a]]\}$
4. $\mathcal{E}[(\text{RESTRICT } a\ t)] = \{\langle x,y\rangle \in \mathcal{D} \times \mathcal{D} \mid \langle x,y\rangle \in \mathcal{E}[a] \text{ and } y \in \mathcal{E}[t]\}$

Finally, for any two types t_1 and t_2, we can say that t_1 *is subsumed by* t_2 if and only if for any set $\mathcal{D}$ and any extension function $\mathcal{E}$ over $\mathcal{D}$, $\mathcal{E}[t_1] \subseteq \mathcal{E}[t_2]$. That is, one type is subsumed by a second type when all instances of the first—in all extensions—are also instances of the second. From a semantic point of view, subsumption dictates a kind of necessary set inclusion.

Given a precise definition of subsumption, we can now consider algorithms for calculating subsumption between descriptions. Intuitively, this seems to present no real problems. To determine if s subsumes t, what we have to do is make sure that each component of s is "implied" by some component (or componens) of t. Moreover, the type of "implication" we need should be fairly simple since $\mathcal{FL}$ has neither a negation nor a disjunction operator.

Unfortunately, such intuitions can be nastily out of line. In particular, let us consider a slight variant of $\mathcal{FL}$—call it $\mathcal{FL}^-$. $\mathcal{FL}^-$ includes all of $\mathcal{FL}$ except for the RESTRICT operator. On the surface, the difference between $\mathcal{FL}^-$ and $\mathcal{FL}$ seems expressively minor.[36] But it turns out that it is computationally very significant. In particular, we have found an $O(n^2)$ algorithm for determining subsumption in $\mathcal{FL}^-$, but have proven that the same problem for $\mathcal{FL}$ is intractable. In the rest of this section, we sketch the form of our algorithm for $\mathcal{FL}^-$ and the proof that subsumption for $\mathcal{FL}$ is as hard as testing for propositional tautologies, and therefore most likely unsolvable in polynomial time. A more formal version of the algorithm and the full proofs can be found in the Appendix.

4.7. Subsumption algorithm for $\mathcal{FL}^-$

1. Flatten both arguments s and t by removing all nested

[35] See Hayes (1979) for some further discussion of the import of languages like KRL. As it turns out, the universal/existential distinction is most often moot, because most frame languages allow only single-valued slots. Thus the slot's meaning is reduced to a simple predication on a single-valued function (e.g., the slot/value pair age:integer means *integer*(*age*(*x*))).

[36] It is the case, however, that there are concepts that can be expressed in $\mathcal{FL}$ that cannot be expressed in $\mathcal{FL}^-$, such as the concept of a person with at least one son and at least one daughter: (AND (SOME (RESTRICT *child male*)) (SOME (RESTRICT *child female*))). In $\mathcal{FL}^-$ all attributes are primitive, so sons and daughters cannot play the same role (child) and yet be distinguished by their types.

AND operators. So, for example,

(AND x (AND y z) w) becomes (AND x y z w)

2. Collect all arguments to an ALL for a given attribute. For example,

(AND (ALL a (AND u v w)) x (ALL a (AND y z)))

becomes

(AND x (ALL a (AND u v w y z)))

3. Assuming s is now (AND $s_1 \ldots s_n$) and t is (AND $t_1 \ldots t_m$), then return T iff for each s_i,
(a) if s_i is an atom or a SOME, then one of the t_j is s_i.
(b) if s_i is (ALL a x), then one of the t_j is (ALL a y), where x subsumes y, calculated recursively.

This algorithm can be shown to compute subsumption correctly (see Appendix A.2.2, Lemma 10). For the purposes of this paper, the main property of the algorithm that we are interested in is that it can be shown to calculate subsumption for $\mathcal{FL}^-$ in $O(n^2)$ time (where n is the length of the longest argument, say). This can be shown roughly as follows (see Appendix A.2.1, Lemma 9 for details): Step 1 can be done in linear time. Step 2 might require a traversal of the expression for each of its elements, and Step 3 might require a traversal of t for each element of s, but both of these can be done in $O(n^2)$ time.

We now turn our attention to the subsumption problem for full $\mathcal{FL}$. The proof that subsumption of descriptions in $\mathcal{FL}$ is intractable is based on a correspondence between this problem and the problem of deciding whether a sentence of propositional logic is implied by another. Specifically, we define a mapping π (see Appendix A.1) from propositional sentences in conjunctive normal form to descriptions in $\mathcal{FL}$ that has the property that for any two sentences α and β, α logically implies β iff $\pi[\alpha]$ is subsumed by $\pi[\beta]$. π itself can be calculated quickly.

What this mapping provides is a way of answering questions of implication by first mapping the two sentences into descriptions in $\mathcal{FL}$ and then seeing if one is subsumed by the other. Moreover, because π can be calculated efficiently, any *good* algorithm for subsumption becomes a good one for implication.

The key observation here, however, is that there can be no [illegible]d algorithm for implication. To see this, note that a sentence implies $(p \wedge \neg p)$ just in case it is not satisfiable. But determining the satisfiability of a sentence in this form is NP-complete (Cook 1971). Therefore, a special case of the implication problem (where the second argument is $(p \wedge \neg p)$) is the complement of an NP-complete one. The correspondence between implication and subsumption, then, leads to the observation that subsumption for $\mathcal{FL}$ is co-NP hard. In other words, since a good algorithm for subsumption would lead to a good one for implication, subsumption over descriptions in $\mathcal{FL}$ is intractable.[37]

5. Conclusions and morals

In this final section, we step back from the details of the specific representational formalisms we have examined and attempt to draw a few conclusions.

An important observation about these formalisms is that we cannot really say that one is *better* than any other; they simply take different positions on the tradeoff between expressiveness and tractability. For example, full FOL is both more expressive and less appealing computationally than a language in semantic-net form. Nor is it reasonable to say that expressiveness is the primary issue and that the other is "merely" one of efficiency. In fact, we are not really talking about efficiency here at all; that, presumably, is an issue of algorithm and data structure, concerns of the Symbol Level (Newell 1981). The tractability concern we have here is much deeper and involves whether or not it makes sense to even think of the language as computationally based.

From the point of view of those doing research in KR, this has a very important consequence: We should continue to design and examine representation languages, *even when these languages can be viewed as special cases of FOL*. What really counts is for these special cases to be interesting both from the point of view of what they can represent, and from the point of view of the reasoning strategies they permit. All of the formalisms we have examined above satisfy these two requirements. To dismiss a language as *just* a subset of FOL is probably as misleading as dismissing the notion of a context-free grammar as just a special case of a context-sensitive one.

What truth in advertising does require, however, is that these special cases of FOL be identified as such. Apart from allowing a systematic comparison of representation languages (as positions on the tradeoff), this might also encourage us to consider systems that use more than one sublanguage and reasoning mechanism (as suggested for equality in Nelson and Oppen (1979)). The KRYPTON language (Brachman *et al.* 1983, 1985), for example, includes all of FOL *and* a frame-description language. To do the necessary reasoning, the system contains both a theorem prover and a description subsumption mechanism, even though the former could do the job of the latter[38] (but much less efficiently). The trick with these *hybrid systems* is to factor the reasoning task so that the specialists are able to cooperate and apply their optimized algorithms without interfering with each other.

These considerations for designers of representation languages apply in a similar way to those interested in populating a KB with a theory of some sort. A good first step might be to write down a set of first-order sentences characterizing the domain, but it is somewhat naive to stop there and claim that the account could be made computational after the fact by the inclusion of a theorem prover and a few well-chosen heuristics. What is really needed is the (much more difficult) analysis of the logical form of the theory, keeping the tradeoff clearly in mind. An excellent example of this is the representation of *time* described in Allen (1983). Allen is very careful to point out what kind of information about time cannot be represented in his system, as well as the computational advantage he gains from this limitation.

It should be noted here that we have addressed only one approach to dealing with the tradeoff. While the tactic of limiting the form of a representation[39] seems to account for almost all current practice in knowledge representation, other ways of

[37] As mentioned in Section. 3.2, the co-NP-complete problems are strongly believed to be unsolvable in polynomial time.

[38] This is true only to a certain extent. See Footnote 9, Brachman and Levesque (1982), and Brachman *et al.* (1983, 1985).

[39] Note that "restricting the form" does not confine us to only simple, obvious types of restrictions. Useful forms of limited languages may have no obvious syntactic relationship to standard logical languages.

avoiding undue complexity should be considered. Especially worthy of attention are weaker logics, having expressive languages but limited power to make inferences (see, for example, Patel-Schneider (1985, 1986) and Levesque (1986)). Another tack to take is to use assumptions as much as possible to produce a tractable, "vivid" knowledge base (Levesque 1986) (being subsequently prepared to undo the effects of assumptions that turn out to be unwarranted).

Finally, one should be aware that the issues addressed here are significant only when concerned with serious scaling up of representations. If there are only a small number of complex sentences (i.e., involving only a small amount of incompleteness), then the tradeoff is a manageable issue. Here we are looking toward representation systems capable of rivalling current database management systems in the number of items stored.

For the future, we still have a lot to learn about the tradeoff. It would be very helpful to accumulate a wide variety of data points involving tractable and intractable languages.[40] Especially significant are crossover points where small changes in a language change its computational character completely (such as that illustrated in Sect. 4.6). Moreover, we need to know more about what *people* find easy or hard to handle. There is no doubt that people can reason when necessary with radically incomplete knowledge (such as that expressible in full FOL) but apparently only by going into a special problem-solving or logic puzzle mode. In normal commonsense situations, when reading a geography book, for instance, the ability to handle disjunctions (say) seems to be quite limited. The question is what forms of incomplete knowledge *can* be handled readily, given that the geography book is not likely to contain any procedural advice on how to reason.

In summary, we feel that there are many interesting issues to pursue involving the tradeoff between expressiveness and tractability. Although there has always been a temptation in KR to set the sights either too low (and provide only a data-structuring facility with little or no inference) or too high (and provide a full theorem-proving facility), this paper argues for the rich world of representation that lies between these two extremes. We should not despair that no matter what we try to do we are faced with intractability, but rather move ahead with the investigation of ways to integrate limited forms of languages and reasoning, with the goal of forging a powerful system out of tractable parts.

Acknowledgements

Many of the ideas presented here originally arose in the context of the KRYPTON project (undertaken mainly at the AI Lab of Schlumberger Palo Alto Research). We are indebted to SPAR for making this research possible and to Richard Fikes, Peter Patel-Schneider, and Victoria Gilbert for major contributions. We would especially like to thank Peter, and also Jim des Rivières and David Etherington for providing very helpful comments on drafts of this paper, and S. J. Hurtubise for not.

ALLEN, J. 1983 Maintaining knowledge about temporal intervals. Communications of the ACM, **26**: 832–843.

BRACHMAN, R. J. 1983. What IS-A is and isn't: an analysis of taxonomic links in semantic networks. IEEE Computer, **16**(10): 30–36.

——— 1985. I Lied about the Trees. AI Magazine, **6**(3): 80–93.

BRACHMAN, R. J., and LEVESQUE, H. J. 1982. Competence in knowledge representation. Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, pp. 189–192.

BRACHMAN, R. J., and SCHMOLZE, J. G. 1985. An overview of the KL-ONE knowledge representation system. Cognitive Science, **9**(2): 171–216.

BRACHMAN, R. J., FIKES, R. E., and LEVESQUE, H. J. 1983. Krypton: a functional approach to knowledge representation. IEEE Computer, **16**(10): 67–73.

BRACHMAN, R. J., GILBERT, V. P., and LEVESQUE, H. J. 1985. An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON. Proceedings of the International Joint Conference on Artificial Intelligence 1985, Los Angeles, CA, pp. 532–539.

COLLINS, A. M., and LOFTUS, E. F. 1975. A spreading-activation theory of semantic processing. Psychological Review, **82**: 407–428.

COOK, S. A. 1971. The complexity of theorem-proving procedures. Proceedings of the 3rd Annual ACM Symposium on Theory of Computing. Association for Computing Machinery, New York, NY, pp. 151–158.

ETHERINGTON, D., and REITER, R. 1983. On inheritance hierarchies with exceptions. Proceedings of the National Conference on Artificial Intelligence, Washington, DC, pp. 104–108.

FRISCH, A., and ALLEN, J. 1982. Knowledge representation and retrieval, for natural language processing. TR 104, Computer Science Department, University of Rochester, Rochester, NY.

GENESERETH, M. R. 1983. An overview of meta-level architecture. Proceedings of the National Conference on Artificial Intelligence, Washington, DC, pp. 119–123.

HAYES, P. J. 1979. The logic of frames. *In* Frame conceptions and text understanding. *Edited by* D. Metzing. Walter de Gruyter and Company, Berlin, West Germany, pp. 46–61.

LEVESQUE, H. J. 1982. A formal treatment of incomplete knowledge bases. Technical Report No. 3, Fairchild Laboratory for Artificial Intelligence Research, Palo Alto, CA.

——— 1984*a*. The logic of incomplete knowledge bases. *In* On conceptual modelling: perspectives from artificial intelligence, databases, and programming languages. *Edited by* M. L. Brodie, J. Mylopoulos, and J. Schmidt. Springer-Verlag, New York, NY, pp. 165–186.

——— 1984*b*. Foundations of a functional approach to knowledge representation. Artificial Intelligence, **23**: 155–212.

——— 1984*c*. A logic of implicit and explicit belief. Proceedings of the National Conference on Artificial Intelligence, Austin, TX, pp. 198–202.

——— 1986. Making believers out of computers. Artificial Intelligence, **30**: 81–108.

LIFSCHITZ, V. 1985. Computing circumscription. Proceedings of the International Joint Conference on Artificial Intelligence 1985, Los Angeles, CA, pp. 121–127.

MOORE, R. C. 1980. Reasoning about knowledge and action. Technical Note 191, SRI International, Menlo Park, CA.

——— 1982. The role of logic in knowledge representation and commonsense reasoning. Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA, pp. 428–433.

NELSON, G., and OPPEN, D. C. 1979. Simplification by cooperating decision procedures. ACM Transactions on Programming Languages and Systems, **1**: 245–257.

NEWELL, A. 1981. The knowledge level. AI Magazine, **2**(2): 1–20.

PATEL-SCHNEIDER, P. F. 1985. A decidable first-order logic for knowledge representation. Proceedings of the International Joint Conference on Artificial Intelligence 1985, Los Angeles, CA, pp. 455–458.

——— 1986. A four-valued semantics for frame-based description languages. Proceedings of the National Conference on Artificial

[40] Some progress has been made on this already at the University of Toronto (Gullen, A., M.Sc. thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, in preparation).

Intelligence, Philadelphia, PA, pp. 344–348.

QUILLIAN, M. R. 1968. Semantic memory. *In* Semantic Information Processing. *Edited by* M. Minsky. MIT Press, Cambridge, MA. pp. 227–270.

REITER, R. 1978*a*. On reasoning by default. Proceedings of Theoretical Issues in Natural Language Processing-2, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, pp. 210–218.

——— 1978*b*. On closed world data bases. *In* Logic and Data Bases. *Edited by* H. Gallaire and J. Minker. Plenum Press, New York, NY, pp. 55–76.

——— 1980. A logic for default reasoning. Artificial Intelligence, **13**: 81–132.

SHAPIRO, S. C. 1979. The SNePS semantic network processing system. *In* Associative networks: representation and use of knowledge by computers. *Edited by* N. V. Findler. Academic Press, New York, NY, pp. 179–203.

SMITH, B. C. 1982. Reflection and semantics in a procedural language. Ph.D. thesis and Technical Report MIT/LCS/TR-272, MIT, Cambridge, MA.

SMITH, D. E., and GENESERETH, M. R. 1985. Ordering conjunctive queries. Artificial Intelligence, **26**: 171–215.

STICKEL, M. E. 1984. A Prolog technology theorem prover. Proceedings of the 1984 Symposium on Logical Programming, Atlantic City, NJ, pp. 211–217.

——— 1985. Automated deduction by theory resolution. Proceedings of the International Joint Conference on Artificial Intelligence 1985, Los Angeles, CA, pp. 1181–1186.

TOURETZKY, D. 1986. The mathematics of inheritance systems. Morgan Kaufmann Publishers, Inc., Los Altos, CA.

WINKER, S. 1982. Generation and verification of finite models and counterexamples using an automated theorem prover answering two open questions. Journal of the ACM, **29**: 273–284.

WOS, L., WINKER, S., SMITH, B., VEROFF, R., and HENSCHEN, L. 1984. A new use of an automated reasoning assistant: open questions in equivalential calculus and the study of infinite domains. Artificial Intelligence, **22**: 303–356.

Appendix: proofs

In this appendix, we present the details of the proofs of the complexity of subsumption for the languages $\mathcal{FL}$ and $\mathcal{FL}^-$. First, we treat the intractability of $\mathcal{FL}$ by showing a direct relation between subsumption in the language and satisfiability in propositional logic. Because propositional satisfiability is difficult, we determine that there can be no good algorithm for subsumption in $\mathcal{FL}$. Subsequently, we show that there exists a sound and complete algorithm for computing subsumption in $\mathcal{FL}^-$ that operates in $O(n^2)$ time.

A.1. The intractability of subsumption for $\mathcal{FL}$

We prove the intractability of subsumption for $\mathcal{FL}$ by showing that for any two propositional formulas α and β, in conjunctive normal form (CNF), there are types in $\mathcal{FL}$, $\pi[\alpha]$, and $\pi[\beta]$ such that

$\models (\alpha \supset \beta)$ iff $\pi[\alpha]$ is subsumed by $\pi[\beta]$

where $\pi[\alpha]$ is roughly the same length as α. Thus, a good algorithm for subsumption in $\mathcal{FL}$ would imply a good algorith for CNF implication. However, CNF implication is difficult, and thus there can be no good algorithm for subsumption.

We being our proof with a lemma relating CNF implication to propositional satisfiability.

Lemma 1

CNF implication is co-NP-hard.

Proof

$\models (\alpha \supset [p \wedge \neg p])$ iff α is unsatisfiable, since otherwise, if $v(\alpha) = \mathrm{T}$ and $\models (\alpha \supset [p \wedge \neg p])$ then $v(p \wedge \neg p)$ would have to be T, which is impossible. But determining if α is satisfiable is NP-hard (Cook 1971). So, determining if $\models (\alpha \supset [p \wedge \neg p])$ is co-NP-hard. Since determining if $\models (\alpha \supset [p \wedge \neg p])$ is a special case of determining if $\models (\alpha \supset \beta)$, then the latter (CNF implication) is co-NP-hard. ■

Next we define the mapping, π, which takes formulas of propositional logic in CNF into types of $\mathcal{FL}$. We assume throughout that clauses do not use the special propositional letters *SELF* or *BOTTOM*.

For any clause c, where $c = (p_1 \vee \ldots \vee p_n \vee \neg p_{n+1} \vee \ldots \vee \neg p_{n+k})$, define $\pi[c]$ =

(AND (ALL (RESTRICT *SELF* p_1) *BOTTOM*)
  (ALL (RESTRICT *SELF* p_n) *BOTTOM*)
  (SOME (RESTRICT *SELF* p_{n+1}))
  ...
  (SOME (RESTRICT *SELF* p_{n+k})))

For any well-formed formula (wff) α in CNF, where $\alpha = (c_1 \wedge \ldots \wedge c_m)$, define $\pi[\alpha]$ =

(AND (ALL (RESTRICT *SELF* (SOME (RESTRICT
    SELF BOTTOM))) *BOTTOM*)
  (ALL (RESTRICT *SELF* $\pi[c_1]$) *BOTTOM*)
  ...
  (ALL (RESTRICT *SELF* $\pi[c_m]$) *BOTTOM*))

Note that for simplicity we will treat single clauses as degenerate conjunctions (so that a wff that is a single clause will be mapped by π into a type that begins with (AND (ALL (RESTRICT *SELF* (SOME (RESTRICT *SELF BOTTOM*))) *BOTTOM*) ...).

Before giving an example of the mapping π, let us introduce a notational convention: Let (NEG p) stand for (ALL (RESTRICT *SELF* p) *BOTTOM*) and (POS p) stand for (SOME (RESTRICT *SELF* p)). So, if, for example, $\alpha = (p \vee q \vee \neg r) \wedge (s \vee \neg t) \wedge (u \vee v)$, then $\pi[a]$ is

(AND (NEG (POS *BOTTOM*))
  (NEG (AND (NEG p) (NEG q) (POS r)))
  (NEG (AND (NEG s) (POS t)))
  (NEG (AND (NEG u) (NEG v)))).

A.1.1. From subsumption to implication

Now, define the following extension function, $\mathcal{E}_0$, over a domain, $\mathcal{D}_0$:

$\mathcal{D}_0$ = [LETTERS → {T,F}] (that is, the domain is all functions that take propositional letters into T and F).
$\mathcal{E}_0[BOTTOM] = \{\ \}$
$\mathcal{E}_0[SELF] = \{\langle v, v\rangle \mid v \in \mathcal{D}_0\}$.
$\mathcal{E}_0[p] = \{v \mid v(p) = \mathrm{T}\}$.

Lemma 2

(a) $v(p) = \mathrm{T}$ iff $v \in \mathcal{E}_0$[(POS p)].
(b) $v(p) = \mathrm{F}$ iff $v \in \mathcal{E}_0$[(NEG p)].

Proof

(a) $v(p) = \mathrm{T}$ iff $v \in \mathcal{E}_0[p]$ iff
  $\langle v, v\rangle \in \mathcal{E}_0$[(RESTRICT *SELF* p)] iff
  $v \in \mathcal{E}_0$[(POS p)].
(b) $v(p) = \mathrm{F}$ iff $v \notin \mathcal{E}_0[p]$ iff
  $\langle v, v\rangle \notin \mathcal{E}_0$[(RESTRICT *SELF* p)] iff
  $v \in \mathcal{E}_0$[(ALL (RESTRICT *SELF* p) *BOTTOM*)]
   (since $\mathcal{E}_0[BOTTOM] = \{\ \}$)
  iff $v \in \mathcal{E}_0$[(NEG p)]. ■

Lemma 3
For any clause c, and any valuation v, $v(c) = \mathrm{F}$ iff $v \in \mathcal{E}[\pi[c]]$.

Proof
$v(p_1 \vee \ldots \vee p_n \vee \neg p_{n+1} \vee \ldots \vee \neg p_{n+k}) = \mathrm{F}$ iff
$\forall i(1 \leq i \leq n)\, v(p_i) = \mathrm{F}$ and $\forall i(1 \leq i \leq k)\, v(p_{n+i}) = \mathrm{T}$ iff
$\forall i(1 \leq i \leq n)\, v \in \mathcal{E}_0[(\mathrm{NEG}\ p_i)]$ and
$\forall i(1 \leq i \leq k)\, v \in \mathcal{E}_0[(\mathrm{POS}\ p_{n+i})]$ by Lemma 2) iff
$v \in \mathcal{E}_0[(\mathrm{AND}\ (\mathrm{NEG}\ p_1) \ldots (\mathrm{NEG}\ p_n)\ (\mathrm{POS}\ p_{n+1}) \ldots (\mathrm{POS}\ p_{n+k}))]$ iff
$v \in \mathcal{E}_0[\pi[p_1 \vee \ldots \vee p_n \vee \neg p_{n+1} \vee \ldots \vee \neg p_{n+k}]]$. ■

Lemma 4
$\mathcal{E}_0[(\mathrm{NEG}\ (\mathrm{POS}\ \mathit{BOTTOM}))] = \mathcal{D}_0$.

Proof
Let $v \in \mathcal{D}_0$. Then $v \notin \mathcal{E}_0[\mathit{BOTTOM}]$. So $\langle v, v\rangle \notin \mathcal{E}_0[(\mathrm{RESTRICT}\ \mathit{SELF}\ \mathit{BOTTOM})]$, so $v \notin \mathcal{E}_0[(\mathrm{POS}\ \mathit{BOTTOM})]$, so $\langle v, v\rangle \notin \mathcal{E}_0[(\mathrm{RESTRICT}\ \mathit{SELF}\ (\mathrm{POS}\ \mathit{BOTTOM}))]$, so $v \in \mathcal{E}_0[(\mathrm{NEG}\ (\mathrm{POS}\ \mathit{BOTTOM}))]$. ■

Lemma 5
If $\pi[\alpha]$ is subsumed by $\pi[\beta]$ then $\models \alpha \supset \beta$.

Proof
Assume $\pi[\alpha]$ is subsumed by $\pi[\beta]$ and $v(\alpha) = \mathrm{T}$, where v is any valuation. So, for each clause c_i of α, $v(c_i) = \mathrm{T}$. Thus, by Lemma 3, $v \notin \mathcal{E}_0[\pi[c_i]]$. So, $\langle v, v\rangle \notin \mathcal{E}_0[(\mathrm{RESTRICT}\ \mathit{SELF}\ \pi[c_i])]$, and so, $v \in \mathcal{E}_0[(\mathrm{NEG}\ \pi[c_i])]$. Also, by Lemma 4, $v \in \mathcal{E}_0[(\mathrm{NEG}\ (\mathrm{POS}\ \mathit{BOTTOM}))]$. Thus, $v \in \mathcal{E}_0[(\mathrm{AND}\ (\mathrm{NEG}\ (\mathrm{POS}\ \mathit{BOTTOM}))\ (\mathrm{NEG}\ \pi[c_1]) \ldots (\mathrm{NEG}\ \pi[c_m]))]$; that is, $v \in \mathcal{E}_0[\pi[\alpha]]$. Since $\pi[\alpha]$ is subsumed by $\pi[\beta]$, $v \in \mathcal{E}_0[\pi[\beta]]$. So $v \in \mathcal{E}_0[(\mathrm{NEG}\ \pi[d_i])]$, for each clause d_i of β. Thus, $v \notin \mathcal{E}_0[\pi[d_i]]$, and by Lemma 3, $v(d_i) = \mathrm{T}$. Thus $v(\beta) = \mathrm{T}$. So for any v, if $v(\alpha) = \mathrm{T}$ then $v(\beta) = \mathrm{T}$, and so $\models (\alpha \supset \beta)$. ■

A.1.2. From implication to subsumption
Given an extension function $\mathcal{E}$ over $\mathcal{D}$ and an element $d \in \mathcal{D}$, define $v_d \in [\mathrm{LETTERS} \rightarrow \{\mathrm{T,F}\}]$ by

$v_d(p) = \mathrm{T}$ iff $d \in \mathcal{E}[(\mathrm{POS}\ p)]$.

Lemma 6
If $\mathcal{E}$ is an extension function over $\mathcal{D}$ and $d \in \mathcal{D}$ where $d \notin \mathcal{E}[(\mathrm{POS}\ \mathit{BOTTOM})]$, then for any clause c, $v_d(c) = \mathrm{F}$ iff $d \in \mathcal{E}[\pi[c]]$.

Proof
Since $d \notin \mathcal{E}[(\mathrm{POS}\ \mathit{BOTTOM})]$, $\forall d^*\ \langle d, d^*\rangle \in \mathcal{E}[\mathit{SELF}] \Rightarrow d^* \notin \mathcal{E}(\mathit{BOTTOM}]$. So, $\forall d^*\ \langle d, d^*\rangle \in \mathcal{E}[\mathit{SELF}] \Rightarrow d^* \notin \mathcal{E}[p]$ iff $\forall d^*\ \langle d, d^*\rangle \in \mathcal{E}[\mathit{SELF}] \Rightarrow d^* \notin \mathcal{E}[p]$ or $d^* \in \mathcal{E}[\mathit{BOTTOM}]$. Thus, $d \notin \mathcal{E}[(\mathrm{POS}\ p)]$ iff $d \in \mathcal{E}[(\mathrm{NEG}\ p)]$. Now, by definition of v_d, $v_d(p_i) = \mathrm{T}$ iff $d \in \mathcal{E}[(\mathrm{POS}\ p)]$ and so $v_d(p_i) = \mathrm{F}$ iff $d \in \mathcal{E}[(\mathrm{NEG}\ p)]$. So $v_d(p_1 \vee \ldots \vee p_n \vee \neg p_{n+1} \vee \ldots \vee \neg p_{n+k}) = \mathrm{F}$ iff $d \in \mathcal{E}[(\mathrm{AND}\ (\mathrm{NEG}\ p_1) \ldots (\mathrm{NEG}\ p_n)\ (\mathrm{POS}\ p_{n+1}) \ldots (\mathrm{POS}\ p_{n+k}))]$; that is, $d \in \mathcal{E}[\pi[p_1 \vee \ldots \vee p_n \vee \neg p_{n+1} \vee \ldots \vee \neg p_{n+k}]]$. ■

Lemma 7
If $\models (\alpha \supset \beta)$ then $\pi[\alpha]$ is subsumed by $\pi[\beta]$.

Proof
Suppose $\models (\alpha \supset \beta)$ and $\mathcal{E}$ is any extension function over some $\mathcal{D}$. Suppose $x \in \mathcal{E}[\pi[\alpha]]$. Then

1. $x \in \mathcal{E}[(\mathrm{NEG}\ (\mathrm{POS}\ \mathit{BOTTOM}))]$, so if $\langle x, y\rangle \in \mathcal{E}[\mathit{SELF}]$ and $y \notin \mathcal{E}[\mathit{BOTTOM}]$, then $y \notin \mathcal{E}[(\mathrm{POS}\ \mathit{BOTTOM})]$.
2. $x \in \mathcal{E}[(\mathrm{NEG}\ \pi[c_i])]$ for each c_i in α, so if $\langle x, y\rangle \in \mathcal{E}[\mathit{SELF}]$ and $y \notin \mathcal{E}[\mathit{BOTTOM}]$, then $y \notin \mathcal{E}[\pi[c_i]]$.

Let d_i be any clause of β and suppose $y \notin \mathcal{E}[\mathit{BOTTOM}]$ and $\langle x, y\rangle \in \mathcal{E}[\mathit{SELF}]$. By (1), $y \notin \mathcal{E}[(\mathrm{POS}\ \mathit{BOTTOM})]$; by (2), $y \notin \mathcal{E}[\pi[c_i]]$ for every c_i in α, and so by Lemma 6, $v_y(c_i) = \mathrm{T}$. Thus, $v_y(\alpha) = \mathrm{T}$. But $\models (\alpha \supset \beta)$, so $v_y(\beta) = \mathrm{T}$ and so $v_y(d_i) = \mathrm{T}$. Then, by Lemma 6 again, $y \notin \mathcal{E}[\pi[d_i]]$. So, if $\langle x, y\rangle \in \mathcal{E}[\mathit{SELF}]$ and $y \in \mathcal{E}[\pi[d_i]]$, then $y \in \mathcal{E}[\mathit{BOTTOM}]$. Thus $x \in \mathcal{E}[(\mathrm{NEG}\ \pi[d_i])]$ and overall, $x \in \mathcal{E}[\pi[\beta]]$. Since this applies to any $\mathcal{E}$ and any x, $\pi[\alpha]$ is subsumed by $\pi[\beta]$. ■

Theorem 1
Subsumption of $\mathcal{FL}$ is co-NP-hard.

Proof
Consider the special case of determining if $\pi[\alpha]$ is subsumed by $\pi[\beta]$. By Lemmas 5 and 7, this is true iff $\models (\alpha \supset \beta)$. But by Lemma 1, this problem is co-NP-hard. Since the size of the expressions are within a polynomial of each other, the first problem is co-NP-hard as well. ■

A.2. The tractability of subsumption for $\mathcal{FL}^-$
Our proof of the tractability of subsumption for $\mathcal{FL}^-$ will proceed as follows: first we provide an alternative, "flat" form for types of $\mathcal{FL}^-$, which makes the definition of a subsumption algorithm straightforward. We then show that the combination of translation of types into the flat form, coupled with the subsumption algorithm for flat types, yields an algorithm that operates in $O(n^2)$ time. Finally, we prove that the algorithm presented does indeed compute subsumption.

A.2.1. Complexity of the subsumption algorithm
Define a subset of $\mathcal{FL}^-$ as follows:

Definition
A type t of $\mathcal{FL}^-$ is a *flat type* iff it is of the form, (AND $t_1 \ldots t_n$), where each t_i is a *flat factor*. A type is a flat factor iff it is atomic, or of the form (SOME a), or of the from (ALL a t), where t is a flat type. In addition, we assume that $t_i \neq t_j$ for $i \neq j$, and that if t_i = (ALL a u) and t_j = (ALL b v) where $i \neq j$, then $a \neq b$.

Lemma 8
If t is a type, then there is an $O(n^2)$ algorithm that converts t to a flat type t' such that $\mathcal{E}[t] = \mathcal{E}[t']$ for any $\mathcal{E}$, and t' is not longer than t.

Proof
First replace (AND x (AND y) z) by (AND x y z), working from the inside out. This clearly does not change any extensions and can be done in linear time. Next, collect arguments to all ALL types, replacing

(AND . . . (ALL a . . .) . . . (ALL a . . .) . . .)

everywhere by

(AND . . . (ALL a (AND . . . . . .)) . . .),

which requires traversing the type at most once per factor. Moreover, this preserves extensions since

$\mathcal{E}[(\mathrm{AND}\ (\mathrm{ALL}\ a\ t)\ (\mathrm{ALL}\ a\ u))]$
$= \{x \mid \forall y\ \langle x, y\rangle \in \mathcal{E}[a] \Rightarrow y \in \mathcal{E}[t]$, and $\forall y\ \langle x, y\rangle \in \mathcal{E}[a] \Rightarrow y \in \mathcal{E}[u]\}$
$= \{x \mid \forall y\ \langle x, y\rangle \in \mathcal{E}[a] \Rightarrow y \in \mathcal{E}[t]$ and $y \in \mathcal{E}[u]\}$
$= \mathcal{E}[(\mathrm{AND}\ (\mathrm{ALL}\ a\ (\mathrm{AND}\ t\ u)))]$. ■

The algorithm for subsumption given flat types is as follows:

```
SUBS?[(AND $x_1, x_2, \ldots x_n$), (AND $y_1, y_2, \ldots y_m$)]:
  do
    let $i \leftarrow 1$
    let covered $\leftarrow$ true
    while (($i \le n$) $\wedge$ covered)
    do
      let $j \leftarrow 1$
      let found $\leftarrow$ false
      while (($j \le m$) $\wedge \neg$found)
      do
        if $x_i \ne$ (ALL $a$ $t$)
          then found $\leftarrow$ ($x_i = y_j$)
          else found $\leftarrow$ (($y_j$ = (ALL $a$ $u$)) $\wedge$ SUBS?[$t, u$])
        $j \leftarrow j + 1$
      end
      covered $\leftarrow$ found
      $i \leftarrow i + 1$
    end
    return covered
  end
```

By Lemma 8, we now need only consider subsumption for flat types (SUBS?). It should be clear from the body of the procedure SUBS? as defined above, that, for flat factors x_i and y_j,

SUBS?[(AND $x_1 \ldots x_n$), (AND $y_1 \ldots y_m$)] returns T iff
$\forall i\, 1 \le i \le n\; \exists j\, 1 \le j \le m$
if $x_i \ne$ (ALL a t) then $x_i = y_j$
otherwise y_j = (ALL a u) and SUBS?[t, u].

Lemma 9
SUBS?[x, y] runs in $O(|x| \times |y|)$ time.

Proof
By induction on the depth of ALL operators in x:
1. If depth = 0, then for each x_i, we must scan all the y_j looking for equal factors, taking $|x| \times |y|$ steps.
2. Assume true for depth $\le k$.
3. Suppose x has maximum depth $= k + 1$ and let x_i be a factor. If $x_i \ne$ (ALL a t) then as before, we must scan y in $|y|$ steps. Suppose there are l factors (ALL a_i t_i) in x. For each such factor, we must find a corresponding one in y, taking $|y|$ steps, and then call SUBS? recursively. By induction, this can be done in roughly $|t_i| \times |y|$ steps, so the total effort for the l factors is

$$\sum_{i=1}^{l} (|y| + |t_i||y|) = \sum_{i=1}^{l} (|t_i| + 1)\,|y|$$

But $\Sigma_{i=1}^{l} (|t_i| + 1)$ is the total length of these factors, so overall, the procedure is completed in $|x| \times |y|$ steps. ■

A.2.2. *Correctness of the subsumption algorithm*

Now, we move on to the proof that this algorithm indeed calculates subsumption: first we must show that if SUBS?[x, y] is T then x indeed subsumes y (soundness); then we must show the converse (completeness). Before beginning, note that the first two steps of the algorithm do not change the extensions of x and y for any extension function, and so do not affect the correctness of the algorithm.

Informally, to see why the algorithm is sound, suppose that SUBS?[x, y] is T and consider one of the conjuncts of x—call it x_i. Either x_i is among the y_j or it is of the form (All a t). In the latter case, there is a (ALL a u) among the y_j, where SUBS?[t, u]. Then, by induction, any extension of u must be a subset of t's and so any extension of y_j must be a subset of x_i's. So no matter what x_i is, the extension of y (which is the conjunction of all the y_j's) must be a subset of x_i. Since this is true for every x_i, the extension of y must also be a subset of the extension of x. So, whenever SUBS?[x, y] is T, x subsumes y. More formally, we have the following lemma.

Lemma 10 (Algorithm soundness)
If SUBS?[(AND $x_1 \ldots x_n$), (AND $y_1 \ldots y_m$)] = T, then (AND $x_1 \ldots x_n$) subsumes (AND $y_1 \ldots y_m$).

Proof
Suppose SUBS? returns T and let $\mathcal{E}$ be any extension function. We will show that $\forall i,\ 1 \le i \le n,\ \exists j,\ 1 \le j \le m$, such that $\mathcal{E}[y_j] \subseteq \mathcal{E}[x_i]$, by induction on the depth of ALL operators in the x_i factors.
1. Suppose x_i does not contain an ALL; then, since SUBS? returns T, $\exists j$ such that $y_j = x_i$; so $\mathcal{E}[y_j] = \mathcal{E}[x_i]$.
2. Assume true for depth $\le k$.
3. If x_i = (ALL a t) then, since SUBS? returns T, $\exists j$ such that y_j = (ALL a u), where SUBS?[t, u] is T. By induction, t must subsume u, so $\mathcal{E}[u] \subseteq \mathcal{E}[t]$. But then $\mathcal{E}$[(ALL a u)] $\subseteq$ $\mathcal{E}$[(ALL a t)].
Now suppose that, for some t, $t \in \mathcal{E}$[(AND $y_1 \ldots y_m$)]. Then $\forall j,\ 1 \le j \le m,\ t \in \mathcal{E}[y_j]$. By the above, $\forall i,\ 1 \le i \le n,\ \exists j,\ 1 \le j \le m$ such that $\mathcal{E}[y_j] \subseteq \mathcal{E}[x_i]$, so $t \in \mathcal{E}[x_i]$. Thus, $t \in \mathcal{E}$[(AND $x_1 \ldots x_n$)]. Since this holds for any t and any $\mathcal{E}$, x subsumes y. ■

Now, we turn to the completeness of the subsumption algorithm. Here we have to be able to show that anytime SUBS?[x, y] is F, there is an extension function that does not assign x to a superset of what it assigns y (i.e., in some possible situation, a y is not an x). Prior to the proof itself, we set up two lemmas. The formal completeness proof will hinge on our ability in all cases where SUBS? returns F to find a factor that is in the "lower" type but not in the "higher" type. The first lemma (11) allows us to construct an extension function over a domain with a distinguished object d_1 that will be in the extension of every type except for a few critical exceptions. This will be used as a counterexample to subsumption in the proof. The second lemma (12) is used in one of the case analyses in the proof.

Lemma 11
Suppose $\mathcal{E}$ is an extension function over $\mathcal{D}$. Suppose that $d_0, d_1 \in \mathcal{D}$, that Q is a primitive type, S is an attribute, and C is a flat type. Furthermore, suppose $\mathcal{E}$ satisfies

1. $d_0 \in \mathcal{E}[p]$ for every primitive p;
2. $d_1 \in \mathcal{E}(p]$ for every primitive p except perhaps Q;
3. $\langle d_0, d\rangle \in \mathcal{E}[a]$ iff $d = d_0$, for every attribute a;
4. $\langle d_1, d\rangle \in \mathcal{E}[a]$ iff $d = d_0$, for every attribute a except perhaps S;
5. $\langle d_1, d\rangle \in \mathcal{E}[S]$ only if $d \in \mathcal{E}[C]$.

Then, for any flat factor t,
(A) $d_0 \in \mathcal{E}[t]$;
(B) if $t \ne Q$ and $t \ne$ (SOME S), and for any u such that u does not subsume C, $t \ne$ (ALL S u), then $d_1 \in \mathcal{E}[t]$.

Proof
(A) By induction on $|t|$:
1. If t is primitive, then true by (1) above.
2. If t is (SOME a), then $\langle d_0, d_0\rangle \in \mathcal{E}[a]$ by (3), so $d_0 \in \mathcal{E}$[(SOME a)].

3. If t is (ALL a u), then since $\forall d\ \langle d_0,d\rangle \in \mathscr{E}[a] \Rightarrow d = d_0$ by (3), we have that $\forall d\ \langle d_0,d\rangle \in \mathscr{E}[a] \Rightarrow d \in \mathscr{E}[u]$ by induction. So $d_0 \in \mathscr{E}[$(ALL a u)$]$.

(B) By cases on t:

1. If t is primitive, then if $t \neq Q$, $d_1 \in \mathscr{E}[t]$ by (2).
2. If t is (SOME a), then if $a \neq S$, $\langle d_1,d_0\rangle \in \mathscr{E}[a]$ by (4), so $d_1 \in \mathscr{E}[$(SOME a)$]$.
3. If t is (ALL a u) where $a \neq S$, then by (4) $\forall d\ \langle d_1,d\rangle \in \mathscr{E}[a] \Rightarrow d = d_0$. So by part A, $\forall d\ \langle d_1,d\rangle \in \mathscr{E}[a] \Rightarrow d \in \mathscr{E}[u]$. So, $d_1 \in \mathscr{E}[$(ALL a u)$]$.
4. If t is (ALL S u), then by (5), $\forall d\ \langle d_1,d\rangle \in \mathscr{E}[S] \Rightarrow d \in \mathscr{E}[C]$. As long as u subsumes C, then $\forall d\ \langle d_1,d\rangle \in \mathscr{E}[S] \Rightarrow d \in \mathscr{E}[u]$. Thus $d_1 \in \mathscr{E}[$(ALL S u)$]$. ■

Lemma 12

For any flat type t, there is an $\mathscr{E}$, a $\mathscr{D}$, and a $d \in \mathscr{D}$ such that $d \notin \mathscr{E}[t]$. (In other words, no type is "tautologous," i.e., a *summum genus*.)

Proof

Suppose t = (AND $p_1 \ldots p_l$ (SOME a_1) ... (SOME a_m) (ALL b_1 u_1) ... (ALL b_n u_n)). If $l \neq 0$ let $\mathscr{D} = \{0\}$, $\mathscr{E}[p_1] = \emptyset$; then $0 \notin \mathscr{E}[t]$. If $m \neq 0$ let $\mathscr{E}[a_1] = \emptyset$, $\mathscr{D} = \{0\}$; then $0 \notin \mathscr{E}[t]$. Otherwise $n \neq 0$ and by induction $\exists \mathscr{E}^*, \mathscr{D}^*, d^*$ such that $d^* \notin \mathscr{E}^*[u_n]$. Let $\mathscr{D} = \mathscr{D}^* \cup \{0\}$ (assuming $0 \notin \mathscr{D}^*$), $\mathscr{E} = \mathscr{E}^*$ except $\mathscr{E}[b_n] = \mathscr{E}^*[b_n] \cup \{\langle 0,d^*\rangle\}$; since $d^* \neq 0$, $d^* \notin \mathscr{E}[u_n]$. Thus, $0 \notin \mathscr{E}[$(ALL b_n u_n)$]$, so $0 \notin \mathscr{E}[t]$. ■

Lemma 13 (Algorithm completeness)

If SUBS?[(AND $x_1 \ldots x_n$), (AND $y_1 \ldots y_m$)] = F, then (AND $x_1 \ldots x_n$) does not subsume (AND $y_1 \ldots y_m$).

Proof

Since SUBS? returns F, there must be an x_i for which there is no corresponding y_j. Given this, we will show by induction on the depth of ALL operators in x_i, how to define an $\mathscr{E}$, a $\mathscr{D}$, and a $d_1 \in \mathscr{D}$ such that $d_1 \notin \mathscr{E}[x_i]$, but $\forall j$, $1 \leq j \leq m$, $d_1 \in \mathscr{E}[y_j]$. Thus, $\mathscr{E}[$(AND $y_1 \ldots y_m$)$] \not\subseteq \mathscr{E}[$(AND $x_1 \ldots x_n$)$]$, and so the former is not subsumed by the latter.

Case 1. Suppose x_i is a primitive, Q. Define $\mathscr{E}$ and $\mathscr{D}$ by

Let $\mathscr{D} = \{0, 1\}$.

Let $\mathscr{E}[t] = \begin{cases} \{0,1\} & \text{if } t \neq Q \\ \{0\} & \text{otherwise.} \end{cases}$

Let $\mathscr{E}[a] = \{\langle 0,0\rangle, \langle 1,0\rangle\}$.

Let $d_1 = 1$. Clearly, $1 \notin \mathscr{E}[Q]$; thus, $1 \notin \mathscr{E}[x_i]$. Now consider any y_j. Let $d_0 = 0$, C = (AND Q), and S = any attribute. The conditions of Lemma 11 are thus satisfied, so unless y_j is one of the named exceptions to Lemma 11B, we know that $1 \in \mathscr{E}[y_j]$. Moreover, y_j cannot be Q since SUBS? returns F; $1 \in \mathscr{E}[$(SOME S)$]$ since $\langle 1,0\rangle \in \mathscr{E}[S]$; and $1 \in \mathscr{E}[$(ALL S u)$]$ since, by Lemma 11, $0 \in \mathscr{E}[u]$ for any u. Thus, no matter what y_j is, $1 \in \mathscr{E}[y_j]$.

Case 2. Suppose x_i is (SOME S). Define $\mathscr{E}$ and $\mathscr{D}$ by

Let $\mathscr{D} = \{0, 1\}$.

Let $\mathscr{E}[t] = \{0, 1\}$.

Let $\mathscr{E}[a] = \begin{cases} \{\langle 0,0\rangle, \langle 1,0\rangle\} & \text{if } a \neq S \\ \{\langle 0,0\rangle\} & \text{otherwise.} \end{cases}$

Again, let $d_1 = 1$, so that $1 \notin \mathscr{E}[x_i]$. Now consider any y_j. Let $d_0 = 0$, C = any flat type, and Q = any primitive. Again, the conditions of Lemma 11 are satisfied, so unless y_j is one of the named exceptions, we know that $1 \in \mathscr{E}[y_j]$. Moreover, $1 \in \mathscr{E}[Q]$, y_j cannot be (SOME S) since SUBS? returns F, and $1 \in \mathscr{E}[$(ALL S u)$]$ for any u. Thus, no matter what y_j is, $1 \in \mathscr{E}[y_j]$.

Case 3. Suppose x_i = (ALL S u), but there is no (ALL S v) among the y_j. By Lemma 12, there are $\mathscr{E}^*$, $\mathscr{D}^*$, d^* such that $d^* \notin \mathscr{E}^*[u]$. Define

$\mathscr{D} = \mathscr{D}^* \cup \{0, 1\}$ (assuming 0 and 1 do not appear in $\mathscr{D}^*$).

$\mathscr{E}[t] = \mathscr{E}^*[t] \cup \{0, 1\}$.

$\mathscr{E}[a] = \mathscr{E}^*[a] \cup \begin{cases} \{\langle 0,0\rangle, \langle 1,0\rangle\} & \text{if } a \neq S \\ \{\langle 0,0\rangle, \langle 1,d^*\rangle\} & \text{if } a = S. \end{cases}$

As before, let $d_1 = 1$. Since $d^* \notin \{0, 1\}$ and $d^* \notin \mathscr{E}^*[u]$, $d^* \notin \mathscr{E}[u]$. So, $1 \notin \mathscr{E}[x_i]$. Now consider any y_j. Let Q be any primitive type and C be any flat type. Again, the conditions of Lemma 11 are satisfied. Moreover, $1 \in \mathscr{E}[Q]$, $1 \in \mathscr{E}[$(SOME S)$]$, and $y_j \notin$ (ALL S v) for any v. So no matter what y_j is, $1 \in \mathscr{E}[y_j]$.

Case 4. Suppose x_i = (ALL S u) and some y_j = (ALL S v) but SUBS?[u, v] = F. By induction, there is a $\mathscr{D}^*$ and an $\mathscr{E}^*$ and a $d^* \in \mathscr{D}^*$ such that $d^* \in \mathscr{E}^*[v]$ but $d^* \notin \mathscr{E}^*[u]$.

Let $\mathscr{D} = \mathscr{D}^* \cup \{0, 1\}$ (assuming 0 and 1 do not appear in $\mathscr{D}^*$).

Let $\mathscr{E}[t] = \mathscr{E}^*[t] \cup \{0, 1\}$.

Let $\mathscr{E}[a] = \mathscr{E}^*[a] \cup \begin{cases} \{\langle 0,0\rangle, \langle 1,0\rangle\} & \text{if } a \neq S \\ \{\langle 0,0\rangle, \langle 1,d^*\rangle\} & \text{if } a = S. \end{cases}$

As before, let $d_1 = 1$. As in the previous case, $1 \notin \mathscr{E}[x_i]$. Now consider any y_j. Let Q be any primitive type and $C = v$. Again, the conditions of Lemma 11 are satisfied. Moreover, $1 \in \mathscr{E}[Q]$, $1 \in \mathscr{E}[$(SOME S)$]$, and for any z different from v, and thus for any z that does not subsume v, $y_j \neq$ (ALL S z). So no matter what y_j is, $1 \in \mathscr{E}[y_j]$. ■

Theorem 2

There is an algorithm to calculate subsumption for $\mathscr{FL}^-$ in $O(n^2)$ time.

Proof

By Lemma 8, types can be flattened in $O(n^2)$ time. The procedure SUBS? works on flattened types in $O(n^2)$ time by Lemma 9, and by Lemmas 10 and 13, SUBS? returns T iff its first argument subsumes its second. ■