Making Abduction More Efficient

Douglas Appelt and Jerry R. Hobbs

SRI International
Menlo Park, California

Introduction

The TACITUS system uses a cost-based abduction scheme for finding and choosing among possible interpretations for natural language texts. Ordinary Prolog-style, backchaining deduction is augmented with the capability of making assumptions and of factoring two goal literals that are unifiable (see Hobbs et al., 1988).

Deduction is combinatorially explosive, and since the abduction scheme augments deduction with two more options at each node—assumption and factoring—it is even more explosive. We have been engaged in an empirical investigation of the behavior of this abductive scheme on a knowledge base of nearly 400 axioms, performing relatively sophisticated linguistic processing. So far, we have begun to experiment, with good results, with three different techniques for controlling abduction—a type hierarchy, unwinding or avoiding transitivity axioms, and various heuristics for reducing the branch factor of the search.

The Type Hierarchy

The first example on which we tested the abductive scheme was the sentence

> There was adequate lube oil.

The system got the correct interpretation, that the lube oil was the lube oil in the lube oil system of the air compressor, and it assumed that that lube oil was adequate. But it also got another interpretation. There is a mention in the knowledge base of the adequacy of the lube oil pressure, so it identified that adequacy with the adequacy mentioned in the sentence. It then assumed that the pressure was lube oil.

It is clear what went wrong here. Pressure is a magnitude whereas lube oil is a material, and magnitudes can't be materials. In principle, abduction requires a check for the consistency of what is assumed, and our knowledge base should have contained axioms from which it could be inferred that a magnitude is not a material. In practice, unconstrained consistency checking is undecidable and, at best, may take a long time. Nevertheless, one can, through the use of a type hierarchy, eliminate a very large number of possible assumptions that are likely to result in an inconsistency. We have consequently implemented a module that specifies the types that various predicate-argument positions can take on, and the likely disjointness relations among types. This is a way of exploiting the specificity of the English lexicon for computational purposes. This addition led to a speed-up of two orders of magnitude.

A further use of the type hierarchy speeds up processing by a factor of 2 to 4. The types provide prefiltering of relevant axioms for compound nominal, coercion, and other very general relations. Suppose, for example, that we wish to prove $rel(a, b)$, and we have the two axioms

$$p_1(x, y) \supset rel(x, y)$$
$$p_2(x, y) \supset rel(x, y)$$

Without a type hierarchy we would have to backchain on both of these axioms. If, however, the first of the axioms is valid only when x and y are of types t_1 and t_2, respectively, and the second is valid only when x and y are of types t_3 and t_4, respectively, and a and b have already been determined to be of types t_1 and t_2, respectively, then we need only backchain on the first of the axioms.

There is a problem with the type hierarchy, however. In an ontologically promiscuous notation, there is no commitment in a primed proposition to truth or existence in the real world. Thus, $lube\text{-}oil'(e, o)$ does not say that o is lube oil or even that it exists; rather it says that e is the eventuality of o's being lube oil. This eventuality may or may not exist in the real world. If it does, then we would express this as $Rexists(e)$, and from that we could derive from axioms the existence of o and the fact that it is lube oil. But e's existential status could be something different. For example, e could be nonexistent, expressed as $not(e)$ in the notation, and in English as "The eventuality e of o's being lube oil does not exist," or simply as "o is not lube oil." Or e may exist only in someone's beliefs or in some other possible world. While the axiom

$$(\forall x) pressure(x) \supset \neg lube\text{-}oil(x)$$

is certainly true, the axiom

$$(\forall e_1, x) pressure'(e_1, x) \supset \neg(\exists e_2) lube\text{-}oil'(e_2, x)$$

would not be true. The fact that a variable occupies the second argument position of the predicate *lube-oil'* does not mean it is lube oil. We cannot properly restrict that argument position to be lube oil, or fluid, or even a material, for that would rule out perfectly true sentences like "Truth is not lube oil."

Generally, when one uses a type hierarchy, one assumes the types to be disjoint sets with cleanly defined boundaries, and one assumes that predicates take arguments of only certain types. There are a lot of problems with this idea. In any case, in our work, we are not buying into this notion that the universe is typed. Rather we are using the type hierarchy strictly as a heuristic, as a set of guesses not about what could or could not *be* but about what it would or would not occur to someone to *say*. When two types are declared to be disjoint, we are saying that they are certainly disjoint in the real world, and that they are very probably disjoint everywhere except in certain bizarre modal contexts. This means, however, that we risk failing on certain rare examples. We could not, for example, deal with the sentence, "It then assumed that the pressure was lube oil."

Unwinding or Avoiding Transitivity Axioms

In general, one must exercise a certain discipline in the axioms one writes. At one point, in order to conclude from the sentence

> Bombs exploded at the offices of French-owned firms in Catalonia.

that the country in which the terrorist incident occurred was Spain, we wrote the following axiom:

$$(\forall x, y, z) in(x, y) \wedge partof(y, z) \supset in(x, z)$$

That is, if x is in y and y is a part of z, then x is also in z. The interpretation of this sentence was taking an extraordinarily long time. When we examined the search space, we discovered that it was dominated by this one axiom. We replaced the axiom with several axioms that limited the depth of recursion to three, and the problem disappeared.

In general, one must exercise a certain discipline in the axioms one writes. Which kinds of axioms cause trouble and how to replace them with adequate but less dangerous axioms is a matter of continuing investigation.

Reducing the Branch Factor of the Search

It is always useful to reduce the branch factor of the search for a proof wherever possible. There are several heuristics we have devised so far for accomplishing this.

The first heuristic is to prove the easiest, most specific conjuncts first, and then to propagate the instantiations. For example, in the domain of naval operations reports, words like "Lafayette" are treated as referring to classes of ships rather than to individual ships. Thus, in the sentence

> Lafayette sighted.

"Lafayette" must be coerced into a physical object that can be sighted. We must prove the expression

$$(\exists x, y)\ Lafayette(x) \wedge rel(y, x)$$

The predicate *Lafayette* is true only of the entity *LAFAYETTE-CLASS*. Thus, rather than trying to prove $rel(y, x)$ first, leading to a very explosive search, we try first to prove $Lafayette(x)$. We succeed immediately, and propagate the value *LAFAYETTE-CLASS* for x. We thus have to prove *rel(y,LAFAYETTE-CLASS)*. Because of the type of *LAFAYETTE-CLASS*, only one axiom applies, namely, the one allowing coercions from types to tokens that says that y must be an instance of *LAFAYETTE-CLASS*.

Similar heuristics involve solving reference problems before coercion problems and proving conjuncts whose source is the head noun of a noun phrase before proving conjuncts derived from adjectives.

Another heuristic is to eliminate assumptions wherever possible. We are better off if at any node, rather than having either to prove an atomic formula or to assume it, we only have to prove it. Some predicates are therefore marked as nonassumable. One category of such predicates are the "closed-world predicates", those predicates such that we know all entities of which the predicate is true. Predicates representing proper names, such as *Enterprise*, and classes, such as *Lafayette*, are examples. We don't assume these predicates because we know that if they are true of some entity, we will be able to prove it.

Another category of such predicates is the "schema-related" predicates. In the naval operations domain, the task is to characterize the participants in incidents described in the message. This is done, as described in Section 5.4. A schema is encoded by means of a schema predication, with an argument for each role in the schema. Lexical realizations and other consequences of schemas are encoded by means of schema axioms. Thus, in the jargon of naval operations reports, a plane can splash another plane. The underlying schema is called *Init-Act*. There is thus an axiom

$$(\forall x, y, \ldots) Init\text{-}Act(x, y, attack, \ldots) \supset splash(x, y)$$

Schema-related predicates like *splash* occurring in the logical form of a sentence are given very large assumption costs, effectively preventing their being assumed. The weight associated with the antecedent of the schema axioms is very very small, so that the schema predication can be assumed very cheaply. This forces backchaining into the schema.

In addition, in the naval operations application, coercion relations are never assumed, since this is what drives the use of the type hierarchy.

Factoring also multiplies the size of the search tree wherever it can occur. As explained above, it is a very powerful method for coreference resolution. It is based on the principle that where it can be inferred that two entities have the same property, there is a good possibility that the two entities are identical. However, this is true only for fairly specific properties. We don't want to factor predicates true of many things. For example, to resolve the noun phrase

ships and planes

we need to prove the expression

$$(\exists x, s_1, y, s_2) Plural(x, s_1) \wedge ship(x) \wedge Plural(y, s_2) \wedge plane(y)$$

where *Plural* is taken to be a relation between the typical element of a set and the set itself. If we applied factoring indiscriminately, then we would factor the conjuncts $Plural(x, s_1)$ and $Plural(y, s_2)$, identifying x with y and s_1 with s_2. If we were lucky, this interpretation would be rejected because of a type violation—planes aren't ships. But this would waste time. It is more reasonable to say that very general predicates such as *Plural* provide no evidence for identity.

The type hierarchy, the discipline imposed in writing axioms, and the heuristics for limiting search all make the system less powerful than it would otherwise be, but we implement these techniques for the sake of efficiency. There is a kind of scale, whose opposite poles are efficiency and power, on which we are trying to locate the system. It is a matter of ongoing investigation where on that scale we achieve optimal performance.

References

[1] Hobbs, Jerry R., Mark Stickel, Paul Martin, and Douglas Edwards, 1988. "Interpretation as Abduction", *Proceedings, 26th Annual Meeting of the Association for Computational Linguistics*, pp. 95-103, Buffalo, New York, June 1988.