COGNITIVE SCIENCE 2, 277 282 (1978)

Tarskian Semantics, or No Notation Without Denotation!

DREW MCDERMOTT
Yale University

Tarskian semantics is called "Tarskian" for historical reasons (Tarski, 1936). A more descriptive name would be "systematic denotational semantics," or SD for short. The method is called "denotational" because it specified the meanings of a notation in terms of what its expressions denote. The method is called "systematic" in hopes that the rules that assign meaning are precise enough to support statements and occasionally proofs of interesting properties of the notation.

In a typical predicate calculus, we assign to primitive symbols denotations which consist of objects, functions, or predicates. Then the meanings of more complex expressions are defined by rules which define their meanings in terms of the meanings of their parts. For sentences in such a language, this amounts to specifying the conditions which make any given sentence *true*. That is, the meaning of a sentence is a specification of what would make it denote T and what would make it denote NIL. This specification may thus be thought of as a generalization of an ordinary LISP predicate definition.

For example, we may assign to the predicate symbol PTRANS a predicate which is true only if its first argument has ever caused its second argument to be physically transferred from its third argument to its fourth argument. Then the denotation of (ACTOR x ⇔ PTRANS OBJ y FROM u TO v) should be T just if the denotation of x has ever transferred the denotation of y from the denotation of u to the denotation of v. (This is a long-winded way of writing a typical semantic rule, which maps the syntax of an expression into a denotation systematically. Syntax is not an issue in this paper, but notice that the use of SD does not commit us to any syntax in particular, so long as it is precise.) It is clear that the first argument of the denotation of PTRANS should be an animate agent; its second, a physical object; its third and fourth, places. If we wish to be precise, we must somehow forbid incongruous types to appear in these places, or go on to specify what the denotation (ACTOR x ⇔ PTRANS OBJ y FROM u TO v) is when x, y, u, or v is incongruous.

So far this may seem very fluffy stuff. What have we gained by (apparently) just repeating in the semantic domain what is fairly obvious in the first place? Mainly we have gained a certain *commitment*. By actually pinning ourselves down, for example, to the requirement that a PTRANS expression signifies that a transfer *ever* occurred, we are in a position to pass judgment on the truth of certain inferences, and on systems which license them

The real power of this method appears when we embed the notation in an inference system of some kind. For example, say we require the inference rule, "If (ACTOR x ⇔ PTRANS OBJ y FROM u TO v), then infer (ACTOR y IS (LOC VAL v))." (Where is IS-LOC-VAL construct is to mean that x has at some time been at v). Given an SD interpretation of this notation, we can ask, Is this rule *sound*? That is, is it true that if x ever transfers y to v, y will at some time have been at v? The answer is clearly yes. On the other hand, consider the rule, "If (ACTOR y IS (LOC VAL vl)) and (NOT (EQUAL vl v)) and (NOT (ACTOR x ⇔ PTRANS OBJ y TO v)) then (NOT (ACTOR y IS (LOC VAL v)))." If our intent is to capture the idea that nothing moves without a PTRANS, we have failed, since, by the interpretation we are building, there is no time relation between the hypothesized PTRANS and the statement (NOT (ACTOR y IS (LOC VAL v))). The rule for NOT will map (NOT p) into T just in case p is mapped into NIL, but this will happen only if the denotation of y has never been and never will be at the place denoted by v. So the rule says, "If y has been some place besides v, and some agent x has never transferred y to v, then y has never been at v." This rule is simply false. Of course, this one test does not mean there is no way to express what we want in this language, but it means the obvious way will not work. (And it provides a strong intuitive argument that this particular language requires extension to be able to denote particular times and events, and quantifications over them.)

Sometimes the precise study of semantics does enable us to make generalizations about everything statable or derivable in an inference system. In particular, we would like to know when a system allows us to infer too little or too much. The Holy Grail of this study is a theorem to the effect that, given an intuitively appealing assignment of meanings to the expressions of a system, its inference rules entitles us to infer exactly the true sentences (those with value T), no more and no less. Such a result is called a *completeness theorem.* Often we have to settle for less, and prove only that the inferences allowed by the system are true. This is a proof of *soundness* and *consistency.*

Even when a system is too complex or evolving too fast for these proofs to be available, the application of SD in an informal way can still be valuable. Here the method suggests a strong self-discipline to be applied in considering adding a rule or predicate symbol to the system. This discipline amounts to asking, Does this new construct denote something we can pin down? Is a proposed rule *true*? If we cannot answer this question, we have no way of foreseeing all the interactions of new constructs with old. Even if we must persist in adding a new rule "blindly," this attitude warns us to be on our guard.

This advice may seem vacuous, but it has application to real AI systems. For example, it puts a heavy burden on designers of production systems.

These are systems of rules of the form "condition → action," where the condition is to match some memory structure and prescribe an action which changes that structure. If the conditions can be given a precise semantics, and if the actions are always of the form, "infer p," we can give an obvious denotational semantics to the rules, and there will be no loose ends. (MYCIN (Shortliffe, 1976) is like this, more or less.) Unfortunately, these restrictions are not met by many such systems. This means that there is no way to say whether a particular rule is sound, without studying the entire system of which it is a part (and even then it is not clear what sort of statement we would like to make about it).

Consider the AMORD system of de Kleer, Doyle, Steele, and Sussman (1977). This is for the most part a very well-disciplined production system where the rules can be given an SD semantics. Its rules are used uniformly as "forward deduction" rules: a → b is used just to infer b after a has been inferred (Notice that heretofore I have not mentioned inference procedure; in practice one must distinguish between all inferences that are *allowed* and the subset that a particular procedure actually *does*.) What if we want to use the same abstract rule to try to prove a as a subgoal of trying to prove b? We can write this as "b ← a," and define "back-arrow" thus:

```
(q ←p) → ((show q) → (show p))
```

That is, "From q ← p, infer that if it is inferred that q is a goal, infer that p is a goal." Both of these symbols, ← and SHOW, are defined by the user, not the system. SHOW can be used to define other goal-oriented constructs. For example, when the goal of proving a conjunction comes up, it is handled by a rule like this (don't worry too much about understanding it):

```
(show (p & q))
→{(show p)
   (p→{ (show q)
      (q → (p & q))})}
```

which means, apparently, "If you wish to show p & q, then you wish to show p, and if p is concluded, you then wish to show q; if p and q are concluded, you may then infer p & q." I say "apparently" because we have not really given a meaning to SHOW, and hence are in no position to judge the soundness of the conjunction rule. You might think (SHOW p) means, "The system is currently interested in the truth of p," but what does (SHOW (ON X BLOCK1)) mean, when X is a variable? Is X to be thought of as universally quantified? That is, is the system, for all X, interested in the truth of (ON X BLOCK1)? This seems doubtful. For instance, if AMORD were to be used as an insurance-company data base, we might have a rule:

```
(show (health john-doe x))
→ (do (cancel-policy john-doe))
```

meaning, apparently, "If someone is interested in John Doe's health, cancel his policy." (Perverse, but not inconceivable for an insurance company.). But then assertion of (SHOW (HEALTH Z BAD)), meaning, "I am interested in whose health is bad," will trigger the rule and cause John's policy to be cancelled. This is not just unfair, it is unsound.

This does not mean AMORD is worthless. It just means it is unanalyzable at a crucial point. Most of the time the semantics of the system is well-behaved; in fact, it uses a version of the well-understood resolution rule of inference. But there are times when the only thing between the system's user and nonsense is caution on the part of the user not to push the SHOW symbol too far. (Not that he knows how far that is.)

Other examples abound. Any system which consists of undisciplined LISP programs is resting on rules whose soundness (and meaning) are in doubt. A system like KRL (Bobrow & Winograd, 1977), which consists of a splendid edifice of notation with no denotation, is a castle in air. Who can say whether two KRL expressions conflict, for instance? Similar criticisms can be made of KRL's cousins, the semantic networks (see Woods, 1975).

I realize that some AI people are liable to resist these ideas stubbornly. They are likely to ask why we should bother with the immediate goal that an inference system be sound, since in the long run the only criterion for such a system is whether it "works." Further, since inference procedures for practical reasons, are bound, to be incomplete with respect to their host inference systems, why insist that the host systems be complete semantically?

The answer is this: It is not just important that a system be correct; it is also crucial that it be understood. Granted that a practical system will be incomplete, we should be able to say in what ways it is incomplete, and why. (For this reason, in the long run, the study of the complexity of inference procedures will be as important as the study of the semantics of inference systems.) After all, a practical program will never be "finished"; it would be nice to know that whatever fragment of one exists will maintain its integrity as new rules are added or new applications are made of old rules. We would like our programs to be "additive," that is, to be able to assimilate new, correct rules from experts without destroying the correctness of old ones. (At least we would like, as in the MYCIN system, for the correctness of old rules to depend on criteria explicit enough for the system to maintain them (Davis, 1976).)

It would perhaps be surprising for an outsider to learn that computer scientists, in spite of the fact that they study purely formal objects like programs and data structures, have a pronounced "anti-formalist" streak. This arose initially from the painful discovery that even the most formal objects have to be debugged. In AI, it comes from our early experience that only trivia could be formalized. Impressive AI programs have been too complex. However, we should not let this stop us. It may be true that formal theories must always remain the study of ideal cases. This has been true in physics, without causing it any harm. (It is difficult to see how physics could have progressed without the ideal gas.) It is also true that formal inquiry will always depend on an influx of good ideas and urgent requirements from the empirical exploration of practical programs. Large practical programs are, however, likely to collapse under their own weight without a good foundation. The structure of programs like AMORD and MYCIN seems to me revealing: they consist of a secure semantic base and patchwork in the poorly understood areas. They work, but, more important, experience with them tells us how to fill in the gaps, so that the next wave of programs can go forward.

Let me now deal with a few more specific objections to "Tarskian semantics." First, there is the objection that, "People do not carry Tarskian semantics around in their heads, so Tarskian semantics is of no concern to AI researchers interested in the way people do things." The premise here is true, the conclusion false. Even the wildest denotationalist has not claimed that semantics should be located "in the head" of a robot. The semantics is for our use, as a tool in analyzing knowledge representations. Of course, if our goal is to duplicate the human representational system, it is not enough to be systematic; we must also be accurate.

The objection has been made that denotational semantics cannot be the semantics of natural language in all its glory. This may or may not be true (if "denotational" is construed broadly), but has nothing to do with its use as a semantics of internal knowledge structures.

One weakness of systematic denotational semantics as developed so far is that it has been mainly a tool of philosophers and logicians, whose goals are rather different from ours. Much of what they have done is of no interest to us, and questions of burning importance to AI they have left untouched. For example, it is characteristic of all logical systems that adding new axioms to a system leaves all old inferences valid. This property is called "monotonicity" (by Minsky, 1974). There is no way to say, for instance, "If you have car keys and gasoline (or money), and there is no information ruling it out, you may use your car to go distances up to a few hundred miles." (Then the required inference may be blocked by the addition of the axiom, "Someone has stolen your tires.") However, this is no objection to denotational semantics as such, but raises the technical problem *within* denotational semantics of representing "... is not ruled out." Much progress has already been made on the practical side of developing programs that can handle constructions like this, but the underlying theory needs work.

There is another area in which most (but by no means all) logicians' results have been inadequate. I said earlier that the denotation of a proposition was always T or NIL. This is called *extensional* semantics, and is standard for mathematical applications. If our language includes a predicate like "BELIEVES," this is inadequate. Consider a proposition like (BELIEVES

MARY (FAT JOHN)). Clearly, the truth value of this proposition does not depend at all on the truth value of (FAT JOHN). So, if the denotation of a formula is to depend only on the denotations of its parts, formulas will have to denote more abstract entities, and have truth values only indirectly. This is the object of *intensional* semantics (Bressan, 1972).

Systematic semantics is a method for solving representational problems, not a catalogue of solutions. We still do not know how to represent time, space, creation and destruction of individuals, knowledge, individuals made out of liquids (Hayes, 1974), and procedures. We will make faster progress on these problems if we keep semantics in mind.

ACKNOWLEDGMENTS

I wish to thank David Barstow, Eugene Charniak, and Patrick Hayes for ideas and criticism. (I haven't bothered to repeat many complementary remarks on semantics made by Hayes (1977).)

REFERENCES

Bobrow, D. & Winograd, T. An overview of KRL, a knowledge representation language. *Cognitive Science*, 1977, *1*, 3.

Bressan, A. *A general interpreted modal calculus*. New Haven: Yale Univ. Press, 1972.

Davis, R. Applications of meta level knowledge to the construction, maintenance and use of large knowledge bases. Stanford AI Laboratory Memo 283. Palo Alto, California, 1976.

Hayes, P. J. Some problems and non-problems in representation theory. *Proc. AISB*, 1974, *1*, 63.

de Kleer, J. Doyle, J., Steele, G. L. & Sussman, G. J. Explicit control of reasoning. MIT AI Laboratory Cambridge, Mass., 1977 Memo. 427. Also in Proceedings of the conference on AI and Programming Languages, Rochester, New York.

Minsky, M. A framework for representing knowledge. MIT AI Memo 306. Cambridge, Mass., 1974.

Shortliffe, E. H. *Computer-based medical consultations: MYCIN*. New York: American Elsevier, 1976.

Tarski, A Der Wahrheitsbegrifl in den formalisierten Sprachen. *Studia Philos*, 1936, *1*, 261.

Woods, W. What's in a link? In D. G. Bobrow & A. Collins (Eds.), *Representation and understanding*. New York: Academic Press, 1975.