

Appendix B Denotational semantics

It is a very old idea that sentences are built up by two parallel processes: combining words on the page, and combining meanings in the mind. Already in the tenth century the Arabic philosopher Al-Fārābī talked of ‘the imitation of the composition of meanings by the composition of expressions’. As long as people agreed with Al-Fārābī that the meanings wear the trousers, there was no way in for mathematics, because nobody had any idea where to look for the mathematical structure of meanings.

During the period 1850–1930 the idea gradually took root that at least for some artificial languages, we can describe the syntax independently of meanings, and then we can describe how meanings of complex phrases are built up from the meanings of words, using the syntax as a template. In 1933 Tarski showed exactly how to build up a semantics in this way for most formal languages of logic. In 1963 Helena Rasiowa and Roman Sikorski popularised an algebraic version of Tarski’s theory: formal languages are algebras with the rules of syntactic composition as their operations, and we interpret a language by describing a homomorphism from the algebra to a suitable structure (e.g. a boolean algebra).

Around 1970 these ideas spread into two new areas. First, Dana Scott and Christopher Strachey showed how to extend Tarski’s framework to computer languages. Second, Richard Montague and Barbara Partee launched a programme to carry the same ideas over into natural languages. Scott and Strachey advertised their scheme as ‘denotational semantics’, while Partee used

Helena Rasiowa Poland, 1917–1994.
We can handle logic by methods of algebra.



the catchword ‘compositionality’, but the ideas involved had a good deal in common. (Both Scott and Montague had worked closely with Tarski.) The work of these four people has been hugely influential in computer science and in linguistics.

Like many people today, in this book we have taken the view that sentences have an inner framework; we represent it by their parsing trees. We *interpret* sentences by climbing up their parsing trees, and we also work out how to *write* or *speak* them by climbing up their parsing trees. Somebody else can worry about whether parsing trees themselves are really syntactic or semantic.

Our main aim in this appendix is to show how the tree-climbing analyses of this book are related to more algebraic accounts of semantics.

Recall the truth table that described the behaviour of truth function symbols. Here we write it using 1 for truth and 0 for falsehood:

(B.1)

ϕ	ψ	$(\phi \wedge \psi)$	$(\phi \vee \psi)$	$(\phi \rightarrow \psi)$	$(\phi \leftrightarrow \psi)$	$(\neg \phi)$	\perp
1	1	1	1	1	1	0	0
1	0	0	1	0	0		
0	1	0	1	1	0	1	
0	0	0	0	1	1		

One can also read this table as a set of definitions of functions:

(B.2)

		b_\wedge	b_\vee	b_\rightarrow	b_\leftrightarrow
1	1	1	1	1	1
1	0	0	1	0	0
0	1	0	1	1	0
0	0	0	0	1	1

Here b_\wedge is a function which takes two truth values to a truth value; for example, $b_\wedge(1, 1) = 1$ and $b_\wedge(1, 0) = 0$ according to the table. Also b_\neg is a function of one truth value, with $b_\neg(1) = 0$ and $b_\neg(0) = 1$; and b_\perp is the constant function with value 0. The b stands for *boolean function*.

In Section 3.5 we saw how to use a σ -structure A to assign a truth value to a propositional formula ϕ . The structure tells us what truth values to write on the leaves, and the truth tables tell us how to put truth values as we climb up the parsing tree of ϕ . Here are the rules for the assignment, written as a compositional definition.

Let σ be a signature and A a σ -structure. We write A^* for the following compositional definition:

$$(B.3) \quad A(\chi) \circ \chi \quad 0 \circ \perp \quad \begin{array}{c} b_{\neg}(v) \circ \neg \\ | \\ v \circ \end{array} \quad \begin{array}{c} b_{\square}(v, w) \circ \square \\ / \quad \backslash \\ v \circ \quad w \circ \end{array}$$

where χ is a propositional symbol and $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

If ϕ is a formula with parsing tree π , we define $A^*(\phi)$, the *truth value of ϕ at A* , to be $A^*(\pi)$.

Linguists and computer scientists would comment at this point that we are only halfway done. We have said what $A^*(\chi)$ is; but A is not part of the language. We need to assign to χ a *semantic value*, in symbols $|\chi|$, that tells us what the truth value of χ is for each possible structure A . In other words, $|\chi|$ should give us $A^*(\chi)$ as a function of A .

The following notation (called *lambda-notation*) is standard for this purpose. We write $\lambda x(x - y)$ for $x - y$ as a function of x . So, for example,

$$(\lambda x(x - y))(6) = 6 - y.$$

Taking the same idea a step further,

$$(\lambda y \lambda x(x - y))(6)(4) = \lambda x(x - 4)(6) = 6 - 4 = 2$$

whereas $(\lambda x \lambda y(x - y))(6)(4)$ works out as -2 . If C is a σ -structure then

$$(\lambda A(A(p_0)))(C) = C(p_0)$$

so that $\lambda A(A(p_0))$ is the function that takes each σ -structure A to the value that A gives to p_0 .

In this notation we have the following *denotational semantics for propositional logic*. We assume a signature σ , and A ranges over all possible σ -structures.

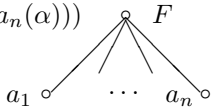
$$(B.4) \quad \lambda A(A(\chi)) \circ \chi \quad 0 \circ \perp \quad \begin{array}{c} \lambda A(b_{\neg}(v(A))) \circ \neg \\ | \\ v \circ \end{array} \quad \begin{array}{c} \lambda A(b_{\square}(v(A), w(A))) \circ \square \\ / \quad \backslash \\ v \circ \quad w \circ \end{array}$$

Applying this compositional definition to the parsing tree of χ , we get $|\chi|$ as the label on the root.

We turn to first-order logic. Just as with boolean functions, we need a more functional notation than we used in the book. Given a structure A , we define an *assignment in A* to be a function α whose domain is a set of variables, which takes each variable to an element of A . If E is an expression whose free variables are all in the domain of α , then we can regard α as a set of instructions for

substituting a name $\overline{\alpha(v)}$ for each free variable v in E . If E is a term, this substitution leads to an element of A , whereas if E is a formula, it leads to a truth value.

We first interpret terms, then formulas. The interpretation of a term t of $\text{LR}(\sigma)$ depends on a σ -structure A and an assignment α in A whose domain includes $FV(t)$. Treating A as fixed, we read off the interpretation of t in A from the following compositional definition:

$$(B.5) \quad \lambda\alpha(\alpha(v)) \circ v \quad \lambda\alpha(c_A) \circ c \quad \lambda\alpha(F_A(a_1(\alpha), \dots, a_n(\alpha)))$$


where v is a variable and F is a function symbol of arity n .

Applying this definition to a term t gets us a function t_A which, applied to an assignment α whose domain includes every variable in t , gives the element of A named by t under this assignment. You can adapt the definition to get $|t| = \lambda A(t_A)$ as a function of the structure A . You might also consider what is needed to make t_A into a function defined only on the assignments whose domain is *exactly* $FV(t)$; but every piece of tidying adds a new layer of clutter to the definition.

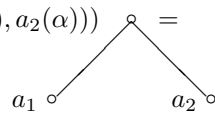
Next we consider relation symbols. If R is an n -ary relation symbol on a set X , its *characteristic function* is the function $\chi_R : X^n \rightarrow \{1, 0\}$ defined by:

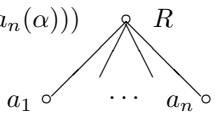
$$\chi_R(a_1, \dots, a_n) = \begin{cases} 1 & \text{if } (a_1, \dots, a_n) \in R \\ 0 & \text{otherwise} \end{cases}$$

Likewise, we define $\chi_=$, the characteristic function of equality, to be the function from X^2 to $\{1, 0\}$ such that

$$\chi_=(a_1, a_2) = \begin{cases} 1 & \text{if } a_1 = a_2 \\ 0 & \text{otherwise} \end{cases}$$

At atomic formulas the denotational semantics has the rules

$$(B.6) \quad \lambda\alpha 0 \circ \perp \quad \lambda\alpha(\chi_=(a_1(\alpha), a_2(\alpha)))$$


$$\lambda\alpha(\chi_R(a_1(\alpha), \dots, a_n(\alpha)))$$


where R is a relation symbol of arity n .

So again the left labels are functions f defined on assignments; but now the values $f(\alpha)$ are truth values (1 or 0).

The definition extends to truth functions by adapting (B.3) to take into account the assignments:

$$(B.7) \quad \begin{array}{ccc} \lambda \alpha b_{\neg}(f(\alpha)) \circ & \neg & \lambda \alpha b_{\square}(f(\alpha), g(\alpha)) \circ \square \\ | & & / \quad \backslash \\ f \circ & & f \circ \quad g \circ \end{array}$$

where $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

It remains only to add clauses for the quantifiers. We write ' $a \in A$ ' as shorthand for ' a is an element of A '. We write $\alpha(a/x)$ for the assignment β defined by

$$\beta(y) = \begin{cases} a & \text{if } y \text{ is } x \\ \alpha(y) & \text{if } \alpha(y) \text{ is defined and } y \text{ is not } x \end{cases}$$

If X is a non-empty subset of $\{0, 1\}$ then $\min X$ and $\max X$ are the minimum and the maximum element of X .

$$(B.8) \quad \begin{array}{ccc} \lambda \alpha \min\{f(\alpha(a/x)) \mid a \in A\} \circ & \forall x & \lambda \alpha \max\{f(\alpha(a/x)) \mid a \in A\} \circ \exists x \\ | & & | \\ f \circ & & f \circ \end{array}$$

where x is a variable.

When we add λA at the front of each left label as before, we create a function that has all σ -structures in its domain, so that its domain is a proper class. This move is legitimate, but it does create some set-theoretic complications. A course in set theory is the place to discuss them.