Inf2A: Introduction to Semantics*

Stuart Anderson

November 17, 2009

Abstract

So far we have concentrated on how to infer a structure for a given sequences of elements (symbols, phonemes, etc). Once we have parsed a sequence of symbols we have a structure for the sequence (i.e. a parse tree). The next question concerns the use of the structure. This note introduces the idea of *semantics*. Semantics considers the "meaning" of a sequence of symbols by providing a mapping (often called a semantic function) that maps from the structure to some other structure, often some abstract mathematical structure where we can reason about the meaning of the sentence. We consider a couple of small examples and some of the issues that arise in defining the semantic mapping.

So far in this course we have concentrated on taking a linear representation (e.g. a sentence in some language) and providing it with a structure that identifies significant subunits of the sentence. In natural language these subunits are things like noun phrases of verb phrases. In formal languages, (e.g. programming languages) these are things like expressions or commands. In this lecture we consider how, given any structured sentence in our language we provide the means to find an interpretation of the sentence in some other structure. This process is usually called providing a semantics for the sentence and the interpretation of a structured sentence is often called the *meaning* of the sentence.

We begin by considering a very small example, namely binary numbers. This example lets us illustrate all the basic ideas in a very simple setting.

A Very Small Example: Binary Numbers

A semantic definition for a language usually comprises three main components. These are:

Abstract syntax: This part defines the sentences we want to provide semantics for. An abstract syntax is a CFG but usually we do not worry about lexical details nor do we worry whether the syntax is ambiguous or not. Generally we provide one production for each different kind of thing belonging to the same syntactic class and assume that we always consider the intended structure for any particular sentence. In the case of binary numerals this just defines a sequence of binary digits.

Semantic algebras: This specifies the mathematical structure we want to map our syntax into, it comprises some domains (here we can just think of these as sets) together with some operations that are used to construct the valuation functions (see next). In the case of binary numerals we are interested in the value of the sequence of bits interpreted as a natural number so we choose $\mathbb{N}$ as the domain into which we map out values and expect to have the usual operations on $\mathbb{N}$ available.

*This Introduction is based on Dave Schmidt's introduction in *Denotational Semantics*, Allyn and Bacon, 1986.

Valuation functions: The valuation function shows how to map from structured sentences to values in our chosen semantic domain. The functions are defined by providing a rule for each different production in the abstract syntax and there is a valuation function defined for each non-terminal in the abstract syntax. So for each non-terminal we provide the type of the valuation function and then give its definition case by case on the productions of the abstract syntax. In the definition of the valuation function we move between the syntactic world (where thing are derivation trees etc to the mathematical world. To make it clear when something is in the syntax provided by the abstract syntax we enclose syntax in special brackets $[\![$ and $]\!]$ that make it clear when some term belongs to the syntax of the language we are providing semantics for.

Abstract syntax

$$B \in \text{Binary-numeral}$$
$$D \in \text{Binary-digit}$$

$$B \to BB \mid D \quad D \to 1 \mid 0$$

Notice here that our abstract syntax is ambiguous. In this case the particular structure chosen does not make a difference to the value of the valuation function we will define. In some cases the structure is important and different structures for the same sentence will have quite different valuations. This should be familiar from the discussion of natural language.

Semantic algebras

$$\text{Domain: } \mathbb{N} \text{ the natural numbers}$$
$$\text{Operations: } 0, 1, 2, \ldots : \mathbb{N};\ 2^{-} : \mathbb{N} \to \mathbb{N};\ +, \times : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

Here we will map from our syntax into a domain that uses the natural numbers. Usually we introduce the domains we intend to use along with the operations we intend to use so we are defining an *algebra* comprising some domain of values together with operations to combine those values. In this case we map from binary-numerals to a pair (n_1, n_2) of integers, where n_1 is the value of the binary-numeral and n_2 is the length of the binary-numeral in bits.

Valuation functions

$\mathcal{B}$: Binary-numeral $\to \mathbb{N} \times \mathbb{N}$
$\mathcal{D}$: Binary-digit $\to \mathbb{N}$

$$\mathcal{B}[\![D]\!] = (\mathcal{D}[\![D]\!], 1)$$
$$\mathcal{B}[\![B_1 B_2]\!] = (v_1 \times 2^{d_2} + v_2, d_1 + d_2) \text{ where } (v_1, d_1) = \mathcal{B}[\![B_1]\!],\ (v_2, d_2) = \mathcal{B}[\![B_2]\!]$$
$$\mathcal{D}[\![0]\!] = 0$$
$$\mathcal{D}[\![1]\!] = 1$$

Here we have defined two valuation functions, one for each of the syntactic categories in the abstract syntax. For each rule for the syntactic category we need to define the value of the valuation function. To find out the value of a binary-numeral B we apply the valuation function $\mathcal{B}$ to B. The result is a pair (n_1, n_2) where n_1 is the value of B and n_2 is the length in bits of B.

Example: Suppose we consider the binary-numeral 1001, this has many parses but here we consider the following parse: [B [B [B [D 1]][B [D 0]]][B [B [D 0]][B [D 1]]]. Applying the valuation function we have:

$$\begin{aligned}
\mathcal{B}[\![1001]\!] &= (v_1 \times 2^{d_2} + v_2, d_1 + d_2) \text{ where } (v_1, d_1) = \mathcal{B}[\![10]\!],\ (v_2, d_2) = \mathcal{B}[\![01]\!] \\
\mathcal{B}[\![01]\!] &= (v_1 \times 2^{d_2} + v_2, d_1 + d_2) \text{ where } (v_1, d_1) = \mathcal{B}[\![0]\!],\ (v_2, d_2) = \mathcal{B}[\![1]\!] \\
&= (v_1 \times 2^{d_2} + v_2, d_1 + d_2) \text{ where } (v_1, d_1) = (\mathcal{D}[\![0]\!], 1),\ (v_2, d_2) = (\mathcal{D}[\![1]\!], 1) \\
&= (1, 2) \\
\mathcal{B}[\![10]\!] &= (2, 2) \\
\mathcal{B}[\![1001]\!] &= (v_1 \times 2^{d_2} + v_2, d_1 + d_2) \text{ where } (v_1, d_1) = (2, 2),\ (v_2, d_2) = (1, 2 \\
&= (9, 4)
\end{aligned}$$

So we have mapped from the world of strings of symbols into the mathematical world of Natural numbers. This is example is very simple but it ilustrates many of the characteristics of more complex languages.

Compositionality

The definition we have given here is called *compositional*. The definition of the *value* of a sentence is given in terms of the *value* of its subunits. This is important in a variety of ways but most importantly it ensures that we are able to inter-substitute subunits of the same value and be sure that the value of the overall sentence is the same. This is not particularly important in the case of Binary-numerals but it is important in programming languages. Can you think of any bad consequences if this were not the case? In actual fact it is quite often the case that programming languages do not have compositional semantics but in principle it is an essential for any tractable language definition.

Now we can consider a larger example that raises some important issues in semantic definitions. We will just consider these briefly since they will mostly be dealt with in courses you may take in third or fourth year.

A Different Compositional Semantics for Binary Numbers

In the notes on semantics for Natural language we saw the use of the *lambda calculus* to construct semantic domains that are functions. We also saw how this approach could be used to construct compositional interpretations of Natural Language. In this section we revisit this approach and use it to construct an alternative interpretation for our binary numbers:

Semantic algebras

$$\begin{aligned}\text{Domain: } &\mathbb{N} \text{ the natural numbers}\\ \text{Operations: } &0, 1, 2, \ldots : \mathbb{N};\ 2\text{-} : \mathbb{N} \to \mathbb{N};\ +, \times : \mathbb{N} \times \mathbb{N} \to \mathbb{N}\end{aligned}$$

We also assume that we can construct functions over the natural numbers an apply those functions to numbers to get other numbers

Valuation functions

$$\begin{aligned}\mathcal{B} &: \text{Binary-numeral} \to (\mathbb{N} \to \mathbb{N})\\ \mathcal{D} &: \text{Binary-digit} \to \mathbb{N}\end{aligned}$$

$$\begin{aligned}\mathcal{B}[\![D]\!] &= \lambda n.(2n + \mathcal{D}[\![D]\!])\\ \mathcal{B}[\![B_1 B_2]\!] &= \lambda n.(\mathcal{B}[\![B_2]\!](\mathcal{B}[\![B_1]\!](n)))\\ \mathcal{D}[\![0]\!] &= 0\\ \mathcal{D}[\![1]\!] &= 1\end{aligned}$$

Example: Suppose we consider the binary-numeral 1001, this has many parses but here we consider the following parse: [B [B [B [D 1]][B [D 0]]][B [B [D 0]][B [D 1]]]. Applying the valuation function we have:

$$\begin{aligned}\mathcal{B}[\![1001]\!] &= \lambda n.(\mathcal{B}[\![01]\!](\mathcal{B}[\![10]\!](n)))\\ &= \lambda n.(\lambda n.(\mathcal{B}[\![1]\!](\mathcal{B}[\![0]\!](n)))(\lambda n.(\mathcal{B}[\![0]\!](\mathcal{B}[\![1]\!](n))))(n))\\ &= \lambda n.(\mathcal{B}[\![1]\!](\mathcal{B}[\![0]\!](\mathcal{B}[\![0]\!](\mathcal{B}[\![1]\!](n)))))\end{aligned}$$

If we apply this resulting function to 0 we discover the value of the expression is 9.

A pocket calculator

We consider a pocket calculator. We turn it on and off by pressing the ⊙ key. In operation we can type a series of expressions followed by the = key to calculate the result. The calculator has a one number store that keeps the value of the previous calculation that may be accessed by pressing the Prev key. In addition there is a conditional construct $E_1?E_2E_3$ that calculates to the value of E_2 if the value of E_1 is non zero and to the value of E_3 otherwise.

Abstract syntax

$P \in$ Program
$S \in$ Expression-sequence
$E \in$ Expression
$N \in$ Numeral

$P \to \circledcirc S \circledcirc$
$S \to E = S \mid \varepsilon$
$E \to E_1 + E_2 \mid E_1 \times E_2 \mid E_1 - E_2 \mid E_1 \div E_2 \mid (E) \mid$ Prev $\mid N \mid E_1 ? E_2 E_3$
$N \to \ldots$

Semantic algebras

Domain: $\mathbb{B}$ – the booleans (aka truth values)
Operations: *tr*,*fa* : $\mathbb{B}$

Domain: $\mathbb{N}$ – the natural numbers
Operations: $0, 1, 2, \ldots : \mathbb{N};\ +, -, \times, \div : \mathbb{N} \times \mathbb{N} \to \mathbb{N};\ =: \mathbb{N} \times \mathbb{N} \to \mathbb{B}$
normalize : $\mathbb{N} \to \mathbb{N}$; normalize$(0) = 0,$ normalize$(x) = 1, x \neq 0$

Valuation function

$$\begin{aligned}
\mathcal{P} &: \text{Program} \to \mathbb{N}^* \\
\mathcal{P}[\![\circledcirc S \circledcirc]\!] &= \mathcal{S}[\![S]\!](0)
\end{aligned}$$

$$\begin{aligned}
\mathcal{S} &: \text{Expression-sequence} \to \mathbb{N} \to \mathbb{N}^* \\
\mathcal{S}[\![E = S]\!](n) &= \text{let } n' = \mathcal{E}[\![E]\!](n) \text{ in } n'\mathcal{S}[\![S]\!](n') \\
\mathcal{S}[\![\varepsilon]\!](n) &= \varepsilon
\end{aligned}$$

$$\begin{aligned}
\mathcal{E} &: \text{Expression} \to \mathbb{N} \to \mathbb{N} \\
\mathcal{E}[\![E_1 + E_2]\!](n) &= \mathcal{E}[\![E_1]\!](n) + \mathcal{E}[\![E_2]\!](n) \\
\mathcal{E}[\![E_1 - E_2]\!](n) &= \mathcal{E}[\![E_1]\!](n) - \mathcal{E}[\![E_2]\!](n) \\
\mathcal{E}[\![E_1 \times E_2]\!](n) &= \mathcal{E}[\![E_1]\!](n) \times \mathcal{E}[\![E_2]\!](n) \\
\mathcal{E}[\![E_1 \div E_2]\!](n) &= \mathcal{E}[\![E_1]\!](n) \div \mathcal{E}[\![E_2]\!](n) \\
\mathcal{E}[\![E_1 ? E_2 E_3]\!](n) &= \text{let } n' = \text{normalize}(\mathcal{E}[\![E_1]\!](n)) \text{ in } n' \times \mathcal{E}[\![E_2]\!](n) + (1 - n) \times \mathcal{E}[\![E_3]\!](n) \\
\mathcal{E}[\![\text{Prev}]\!](n) &= n \\
\mathcal{E}[\![(E)]\!](n) &= \mathcal{E}[\![E]\!](n) \\
\mathcal{E}[\![N]\!](n) &= \mathcal{N}[\![N]\!]
\end{aligned}$$

Issues

There are a number of important issues that arise in attempting to give a compositional semantics for this language. We do not have time to investigate these in any great detail – but here is a summary.

Exceptions What happens if we try to evaluate $3 \div 0$? More particularly what happens to the following calculation that may or may not use Prev? Generally this can be dealt with in a variety of ways – the simplest approach is to add error elements to the semantic domains. This is a simple fix – but in complex languages it can get out of hand very quickly.

Definedness of the semantic function Sometimes it seems the semantic functions are not exactly "normal" kinds of functions. For example, consider evaluating $1?(3)(3 \div 0)$. What is its value? Is this what we want? Do you have any ideas on how to change the defintiion to achieve the intended meaning of this construction.

Limits to compositional definitions Are there semantic functions that we might be interested in that are *not* compositional? It is the case that there do seem to be some features of programming languages that are hard to capture using compositional definitions but you will not encounter them until later in your career.