

# Introduction to Semantics

Stuart Anderson

School of Informatics  
University of Edinburgh

17 November, 2009

# The Recipe for a Semantic Definition

- Grammars give us a way of mapping from a sequence to a structure (usually a tree structure). The tree structure provides an analysis of the role of different substrings in the sentence we have parsed.
- In natural language this analysis is in terms of parts-of-speech and the role of the sentence in communication. For computer languages the structure is usually describing expressions and commands that could be executed by a computer.

# The Recipe for a Semantic Definition

- Providing a *semantics* for a language involves giving a (systematic) mapping from all possible structures ascribed to sentences in the language to “meaning”.
- For example:
  - For natural language the “meaning” might be a mapping from environments (i.e. partial descriptions of the world) to truth values (or possibly true, false, don't know). The interpretation of such functions might be this utterance is true/false in this environment.
  - For computer languages the semantics of a programming language would be from syntactically correct programs to functions that map from the state of the machine to the state of the machine (or something more complex to take account of I/O).

# The Recipe for a Semantic Definition

- To provide a mapping from our sentence structure to some kind of “meaning” we need three things:
  - *Abstract Syntax* - this is a “clean” description of the structure we are considering. This usually omits fine lexical detail and is not particularly concerned with whether the language is ambiguous or not. It is a description of the possible structures that will be considered rather than a description of how a structure can be derived from a linear presentation.

## The Recipe for a Semantic Definition - Continued

- *Semantic Algebras* - this is a description of the mathematical structure we will use to interpret the structures in the Abstract Syntax. This is almost always some variety of Algebra (i.e. a collection of values plus some operations for combining those values). This is usually abstract with respect to many implementation issues (e.g. we use the natural numbers or real numbers *not* the finite collection of 64-bit numbers or the 64-bit floating-point numbers).
- *Valuation Functions* - these are a description of how to map from the abstract syntax to the Semantic Algebras. These are usually defined on the structure of the Abstract Syntax. In defining these we almost always want to know when we are talking about some piece of syntax – syntax is usually contained inside special brackets:  $\llbracket$  and  $\rrbracket$ .

# The Recipe: Binary Numbers: Abstract Syntax

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

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

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

# The Recipe: Binary Numbers: Semantic Algebras

Domain:  $\mathbb{N}$  the natural numbers

Operations:  $0, 1, 2, \dots : \mathbb{N}$ ;  $2^- : \mathbb{N} \rightarrow \mathbb{N}$ ;  $+, \times : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

# The Recipe: Binary Numbers: Valuation Functions

$\mathcal{B}$  : Binary-numeral  $\rightarrow \mathbb{N} \times \mathbb{N}$

$\mathcal{D}$  : Binary-digit  $\rightarrow \mathbb{N}$

$$\mathcal{B}[[D]] = (\mathcal{D}[[D]], 1)$$

$$\mathcal{B}[[B_1 B_2]] = (v_1 \times 2^{d_2} + v_2, d_1 + d_2)$$

$$\text{where } (v_1, d_1) = \mathcal{B}[[B_1]], (v_2, d_2) = \mathcal{B}[[B_2]]$$

$$\mathcal{D}[[0]] = 0$$

$$\mathcal{D}[[1]] = 1$$

## The Recipe: Binary Numbers: Example

$$\mathcal{B}[\![1001]\!] = (v_1 \times 2^{d_2} + v_2, d_1 + d_2)$$

$$\text{where } (v_1, d_1) = \mathcal{B}[\![10]\!], (v_2, d_2) = \mathcal{B}[\![01]\!]$$

$$\mathcal{B}[\![01]\!] = (v_1 \times 2^{d_2} + v_2, d_1 + d_2)$$

$$\text{where } (v_1, d_1) = \mathcal{B}[\![0]\!], (v_2, d_2) = \mathcal{B}[\![1]\!]$$

$$= (v_1 \times 2^{d_2} + v_2, d_1 + d_2)$$

$$\text{where } (v_1, d_1) = (\mathcal{D}[\![0]\!], 1), (v_2, d_2) = (\mathcal{D}[\![1]\!], 1)$$

$$= (1, 2)$$

$$\mathcal{B}[\![10]\!] = (2, 2)$$

$$\mathcal{B}[\![1001]\!] = (v_1 \times 2^{d_2} + v_2, d_1 + d_2)$$

$$\text{where } (v_1, d_1) = (2, 2), (v_2, d_2) = (1, 2)$$

$$= (9, 4)$$

# Semantics: Compositionality

- This definition has the important property of compositionality.
- I.e. the value of a component is given in terms of the value of the component parts.
- Often we have to manipulate the notion of “value” to achieve this (e.g. in the binary number example we included a measure of the number of bits as well as the numerical value).
- Suppose we had a non-compositional semantics what issues might arise?

# Semantics: Pocket Calculator: Syntax

$P \in \text{Program}$

$S \in \text{Expression-sequence}$

$E \in \text{Expression}$

$N \in \text{Numeral}$

$P \rightarrow \odot S \odot$

$S \rightarrow E = S \mid \varepsilon$

$E \rightarrow E_1 + E_2 \mid E_1 \times E_2 \mid E_1 - E_2 \mid E_1 \div E_2 \mid (E) \mid \text{Prev} \mid N \mid E_1 ? E_2 E_3$

$N \rightarrow \dots$

## Semantics: Pocket Calculator: Semantic Algebras

Domain:  $\mathbb{B}$  – the booleans (aka truth values)

Operations:  $tr, fa : \mathbb{B}$

Domain:  $\mathbb{N}$  – the natural numbers

Operations:  $0, 1, 2, \dots : \mathbb{N}$ ;  $+, -, \times, \div : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ ;  $= : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$

$normalize : \mathbb{N} \rightarrow \mathbb{N}$ ;

$normalize(0) = 0$ ,

$normalize(x) = 1, x \neq 0$

## Semantics: Pocket Calculator: Valuation

$$\mathcal{P} : \text{Program} \rightarrow \mathbb{N}^*$$

$$\mathcal{P}[\![\odot S \odot]\!] = \mathcal{S}[\![S]\!](0)$$

$$\mathcal{S} : \text{Expression-sequence} \rightarrow \mathbb{N} \rightarrow \mathbb{N}^*$$

## Semantics: Pocket Calculator: Valuation

$$\begin{aligned} S[E = S](n) &= \text{let } n' = \mathcal{E}[E](n) \text{ in } n' S[S](n') \\ S[\varepsilon](n) &= \varepsilon \end{aligned}$$

## Semantics: Pocket Calculator: Valuation

$\mathcal{E} : \text{Expression} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$

$$\mathcal{E}[[E_1 + E_2]](n) = \mathcal{E}[[E_1]](n) + \mathcal{E}[[E_2]](n)$$

$$\mathcal{E}[[E_1 - E_2]](n) = \mathcal{E}[[E_1]](n) - \mathcal{E}[[E_2]](n)$$

$$\mathcal{E}[[E_1 \times E_2]](n) = \mathcal{E}[[E_1]](n) \times \mathcal{E}[[E_2]](n)$$

$$\mathcal{E}[[E_1 \div E_2]](n) = \mathcal{E}[[E_1]](n) \div \mathcal{E}[[E_2]](n)$$

$$\mathcal{E}[[E_1?E_2E_3]](n) = \text{let } n' = \text{normalize}(\mathcal{E}[[E_1]](n)) \\ \text{in } n' \times \mathcal{E}[[E_2]](n) + (1 - n') \times \mathcal{E}[[E_3]](n)$$

$$\mathcal{E}[[\text{Prev}]](n) = n$$

$$\mathcal{E}[[\langle E \rangle]](n) = \mathcal{E}[[E]](n)$$

$$\mathcal{E}[[N]](n) = \mathcal{N}[[N]]$$

# Semantics: Pocket Calculator: Issues

- Exceptions: what happens if we try to take account of exceptional conditions (e.g. division by zero)?
- Definedness: are these functions always well-defined for any calculation?
- Limits to compositionality: are there languages for which there is no compositional definition? (there are some that we can't do at the moment)