The Review of Symbolic Logic
Volume 14, Number 3, September 2021

COMPOSITIONALITY, COMPUTABILITY, AND COMPLEXITY

PETER PAGIN
Stockholm University

Abstract. This paper starts from the observation that the standard arguments for compositionality are really arguments for the computability of semantics. Since computability does not entail compositionality, the question of what justifies compositionality recurs. The paper then elaborates on the idea of recursive semantics as corresponding to computable semantics. It is then shown by means of time complexity theory and with the use of term rewriting as systems of semantic computation, that syntactically unrestricted, noncompositional recursive semantics leads to computational explosion (factorial complexity). Hence, with combinatorially unrestricted syntax, semantics with tractable time complexity is compositional.

§1. Introduction. Standard arguments for the claim that natural language meaning is compositional, like the *learnability argument*,[1] take as a premise that language users have the ability to *work out* the meaning of new sentences, i.e. sentences they have never come across before. The fact that meanings of sentences *can* be worked out somehow from what users already know is then associated with the principle of compositionality (informally stated):

(PCI) The meaning of a complex expression is a function of the meanings of its parts and its mode of composition.

The inference from the *working out* premise to the conclusion that (PCI) is true of natural languages is perhaps best seen as an *inference to the best explanation*.[2] The idea is that language users know the meanings of individual words and the semantic significance of syntactic modes of composition (and morphological modifications), and by simply combining knowledge of these things, a user can step by step work out the meaning of new grammatical sentences. That this is possible is supposed to be guaranteed by compositionality.

Received: July 10, 2017.
2020 *Mathematics Subject Classification*: 11Y16, 68Q55.
Key words and phrases: semantic compositionality, recursive semantics, computable semantics, semantic complexity, cognitive difficulty

[1] For the learnability argument, see Davidson (1965) (and for a revised interpretation of that paper, cf. Pagin, 2019b). For an overview and discussion of arguments for compositionality, see Pagin & Westerståhl (2010b).

[2] This characterization is explicit in Fodor (1987) for the systematicity and productivity arguments.

doi:10.1017/S1755020320000027

There are two main problems with arguments of this kind:

(A) That meaning is compositional does not entail that the meaning of new sentences can be worked out.
(B) That the meaning of new sentences can be worked out does not entail that meaning is compositional.

Problem (A) depends on the fact that in (PCI), as well as in more formal versions of the principle, it is not required of the *function* mentioned that it be possible to *work out* its values from the arguments. Problem (B) depends on the fact that when it *is* possible to work out the meanings of a complex expression, what the meaning is may depend on factors over and above the meanings of the parts: then whether we have a part *a* or a part *b* can make a difference to the outcome even if *a* and *b* have the same meaning. In that case, compositionality does not hold.

Clearly, we can deal with problem (A) by simply *adding* the requirement that the values of the function can be worked out. This gives us a principle stronger than the standard principle of compositionality. Problem (B) is more difficult. If the meanings of complex expressions can be worked out in a language with noncompositional semantics, what reason is there, if any, to believe that natural language nevertheless has a compositional semantics? The problem is pressing, since the idea that the meaning of complex expressions is compositional is not only very intuitive, but it also has a long tradition; its roots go back at least to medieval times.[3]

In this paper I shall propose an answer to this question. Briefly, the proposal is that compositionality is associated with *simplicity*, more precisely *computational simplicity*. A *computation* in the technical sense, is a sequence of elementary transformations of representations according to specified rules or patterns that only exploit the syntactic properties of the representations. A computation of the value of a function for a particular argument starts with representation of the argument, and step by step transforms the argument representation into a representation of the value.

As we shall see below, the meaning of an expression is something that can be computed in this sense. We can then speak of *semantic* computation. That semantics is computable does not entail that the comprehension process, when a hearer works out the meaning of an utterance, is a computation in the same sense, even in part. For one thing, this presupposes the so-called *representational theory of mind*, according to which being in a mental state with propositional content consists in tokening a mental representation (cf. Cummins, 1989) with that content, something which is controversial and which I don't need to assume here.

Nevertheless, I shall assume that there is a correlation between the *cognitive difficulty* of a semantic interpretation task and its computational difficulty:

(COG) Other things equal, the greater the computational complexity of an interpretation task, the greater the cognitive difficulty of the interpretation.

[3] The so-called *Additive Principle*, that *the meaning of a complex is the sum of the meanings of the parts*, was stated in an early version already by Abelard in the twelfth century (Abelard, 2010), and in a more mature form, in early middle fourteenth century, by Buridan (Buridan & van Lecq, 1998, 2.3, Soph. 2, thesis 5, QM 5.14, fol. 23vb). This conception appears to have become standard among late medieval logicians.

The general concept of *computational complexity* is the concept of the *difficulty* of a computation, and more precisely of the resources that are required for carrying out the computation (much more below). The (COG) principle does not exclude the possibility that there are other factors, not related to computational complexity, that affect the cognitive difficulty of interpretation. Nevertheless, it is a substantial empirical assumption. I shall leave it as an assumption here.[4] It is a substantial issue in the philosophical background to the present topic, but not directly relevant to the technical matters that will be in focus below.

The cognitive difficulty, or computational complexity, aspect, provides two reasons why the requirement of computability in principle is too weak. The first reason is that there must be an *upper bound* the cognitive difficulty, and hence by (COG), to the computational complexity, of interpretation tasks that humans can discharge. Human cognitive capacities are limited. Attention span, memory, life span, even the number of neurons of a human being, are finite. For all these reasons, there is an upper bound to the size of computations we can perform. Therefore, the claim that natural language meaning is *computable*, or that it can be worked out *in principle*, is too weak for explaining the fact that language users are capable of working out the meaning of new sentences. Since computability as such sets no upper bound to complexity, the meanings of complex expressions of some language L might well be computable, even though it is too difficult for any human speaker to actually work them out. Hence, to have an explanation at all, we need to assume (using (COG)) that meanings are computable by formal methods (algorithms) that are tractable to the human mind. The appeal to computability is too weak already as it stands. We can apply this observation as the requirement on natural language semantics: that natural language meaning be computationally *tractable*:

(TRC) Any plausible semantics for natural language is computationally tractable.

The second reason derives from the observation that semantic comprehension is a very *fast* real-time process. The hearer typically interprets an utterance on-line, as it is being produced, with comprehension achieved immediately after the utterance is completed. Therefore, under the (COG) assumption, the complexity of semantic computation must be *low*. It must be low enough to make it possible for us to process utterances as fast as we do. How low is that? Much more needs to be in place before a sensible answer can even be stated, let alone justified. However, it is reasonable to conjecture that, for semantic theories, other things equal, the lower the better:

(MIN) Other things equal, the lower the computational complexity of a semantics, the better.

(MIN) is not a conceptual truth, as little as (COG) or (TRC). It might well be that because of idiosyncrasies of the human brain, the true semantic theory for a language isn't the one with lowest complexity. This might be because (COG) is false, and even if (COG) is true, idiosyncratic features of language *learning* mechanisms might lead us to a semantics that is in fact not the easiest to process. That is implausible, however, and I believe the reflections below will reinforce this assessment.

[4] I discuss the relevance of computational complexity to cognition in Pagin (2012a), Pagin (2012b), and in Pagin (2013).

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

(COG) together with (TRC) or (MIN) can support an appeal to *the inference to the best explanation*. It may be that some particular kind of semantics *minimizes* semantic computational complexity or makes it tractable. Such a semantics then offers *the best explanation* of real-time linguistic communicative success. By the *best explanation* inference, we conclude that natural language has a semantics of this kind.

How does compositionality fit into this? We noted that compositionality does not entail computability. We will also see that compositionality together with computability does not guarantee low complexity. It will turn out, however, that minimal semantic complexity *does* entail compositionality. It will also turn out that allowing semantics that is not (general) compositional entails allowing semantics with nontractable complexity. These are the main results of this paper. Of course, as just stated, it is only roughly true: a number of important qualifications will be made in later sections. But already at this point we can state intuitively the main answer to the question "Why compositionality?."

The answer is that whenever the meaning of an expression depends on more than one factor, processing the interdependence of these factors needs extra computational steps. So, if the meaning of a complex expression depends on the meanings of the parts and the mode of composition, and on *other* features of the parts as well, more steps will be needed in computing the meaning. If in addition the other features are both complex and interdependent, the complexity will quickly grow. Therefore, since there is anyway dependence on part meanings and mode of composition, and since minimizing complexity entails eliminating any further dependence, what remains is compositional semantics. This, in a nutshell, is the outcome of the technical investigations in this paper.[5]

However, although the basic idea is simple, it is not a simple matter to establish the result with a reasonable degree of precision. The proof that minimally complex semantics is compositional depends on the fact that certain kinds of compositional semantics have minimal complexity in an *absolute* sense, not just less than any other kind of semantics. Now, even with the complexity measure that will be chosen here, minimal complexity can be defined in two different ways. Under strong restrictions on the character of the rule systems that will be investigated, restrictions that do not allow for so-called *speed-up*, minimal complexity will amount to a measure of exactly one computation step per symbol in the syntactic term that is processed. If the object language (OL) contains quantifiers or other variable binding operators, and the semantics has a standard format, then complexity will not be minimal in this sense. If speed-up is allowed, minimal complexity will amount to a complexity function that is a *linear* function of the size of the syntactic term. If the language contains variable-binding, however, the minimal complexity will not be linear but quadratic.[6] Variable-binding in the OL gives rise to a number of complications, and these cannot be given a full treatment in the present paper.

[5] In Pagin (2012a), I have discussed the philosophical aspects of the idea at greater length. This paper focuses on the technical side.

[6] I characterize and discuss the complexity of truth-theoretic semantics for first-order languages in Pagin (2012b).

I shall also leave aside extra-linguistic context dependence. This is not because extra-linguistic context dependence increases computational complexity. On the whole, it presumably does increase cognitive effort on the part of the hearer, but in ways that are quite different from what will be considered here (e.g. the time needed to identify referents of demonstratives). However, I know of no computational model of this cognitive process. For present purposes, taking account of extra-linguistic context dependence adds an inessential complication. I shall take account of certain kinds of *linguistic* context dependence (see §2 on general compositionality).

The rest of the paper is organized as follows. In §2. I set out the formal framework for compositional semantics, including general compositional semantics. §3 discusses computability and recursive semantics. §4 introduces the idea of computational complexity and its application to semantics. §5 discusses the choice of complexity measure. §6 introduces *term rewriting* and its application to semantics. §7 relates term rewriting and complexity measures. §8 presents the main complexity results in terms of a measure inherent to term rewriting. §9 sums up the resulting relation between compositionality and complexity. Appendix A. provides the proof of proposition 1, concerning recursive semantics. Appendix B. provides proof of proposition 6. Appendix C provides the proof of Proposition 7, concerning the complexity of recursive semantics. Appendix D. provides the proof of proposition 8, concerning the termination property of a relevant class of term rewriting systems.

§2. Compositionality. I shall here use a version of the abstract algebraic framework for semantics that was introduced by Hodges (2001) and subsequently used e.g. in Pagin (2003), Westerståhl (2004), Pagin & Westerståhl (2010a), and Pagin & Westerståhl (2010b). This approach uses an algebraic format in setting out the grammar for a language, and lets the *terms* of a grammatical term algebra be the arguments to semantic functions. This framework doesn't use *grammatical sorts* for defining the domains of the syntactic operations. Rather, the operations are simply allowed to be partial functions. This simplifies the abstract treatment.

2.1. Syntax. The syntax for a language L will be given in the format of a *grammatical term algebra* GTA_L.

Definition 1. *A grammatical term algebra GTA_L for a language L is a partial algebra*

$$(GT_L, AT_L, \Sigma_L),$$

where GT_L is the set of grammatical terms for L, AT_L is the set of atomic (grammatical) terms for L, and Σ_L is the set of syntactic operations for L. Σ_L is required to be finite. AT_L is finite as well, unless L contains variable binding operators and therefore a countable set of variables.[7] *GT_L is the closure of AT_L under Σ_L.*

I shall use $s, s_1, s_2, \ldots, t, t_1, t_2, \ldots, u, u_1, u_2, \ldots$ as parameters for grammatical terms. I shall use $\sigma, \sigma_1, \sigma_2, \ldots$ as parameters for syntactic operations (quotes omitted here). An instance of

$$\ulcorner\sigma(t_1, \ldots, t_n)\urcorner$$

[7] This will not be considered here.

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

is a *meta-grammatical* term. It *refers to* a grammatical term in GT_L, provided the corresponding operation σ is defined for the arguments $t_1, \ldots, t_n$. If it does, then $t_1, \ldots, t_n$ are the *immediate subterms* of $\sigma(t_1, \ldots, t_n)$.

What do the grammatical terms themselves look like? This varies from system to system. In Hodges' original setup, there is an *expression algebra*

$$EA_L = (E_L, A_L, \Sigma'_L),$$

where E_L is the set of well-formed *expressions* (strings) of L, A_L is the set of *atomic expressions* of L and Σ'_L is the set of syntactic operations on *expressions* of L. E_L is then the closure of A_L under Σ'_L. The grammatical terms are built up from the atomic terms and the operators (i.e. the symbols denoting the operations of the algebra). Hodges lets atomic expressions double as atomic grammatical terms. Let us for the moment follow Hodges in this. Let 'e_1' and 'e_2' be atomic expressions in E_L, and '$\overline{\sigma}$' a two-place operator, where the operation $\overline{\sigma}$ is in Σ'_L.[8] Then, if $\overline{\sigma}$ is defined for 'e_1' and 'e_2',

$$`\overline{\sigma}(`e_1', `e_2')'$$

is a grammatical term. It represents the expression of L that is formed from 'e_1' and 'e_2' by means of the *operation* $\overline{\sigma}$, i.e. $\overline{\sigma}$('e'_1', 'e'_2').[9] Assume that σ is the corresponding syntactic operation in the grammatical term algebra. Then while the operation $\overline{\sigma}$, in the expression algebra, maps expressions on expressions, the *operation* σ in the grammatical term algebra, maps terms on terms, and more precisely terms on more complex terms that are formed by means of the operator '$\overline{\sigma}$'. That is, we have

$$\sigma(`e_1', `e_2') \quad = \quad `\overline{\sigma}(`e_1', `e_2')'. \tag{1}$$

In Hodges' format, the grammatical terms are both the input to the semantics *and* directly represent the expressions of the language. In other frameworks the relation is not direct. In the format of Heim & Kratzer (1998), the syntax is a version of the *Principles and Parameters* framework. There the inputs to the semantics belong to the LF level of syntactic representation. LF does not directly represent expressions of the language. Rather, it is the *surface structures* that directly represent the expressions, and LF representations are *derived* from surface structures. In Heim & Kratzer (1998), items in LF are presented as syntactic trees. Trees, of course, correspond in the linear format to terms, with subtrees corresponding to subterms.

For present purposes, it does not matter what the syntactic format is like, as long as the items that are the arguments to the semantic functions have the constituent structure corresponding to the subterm relation. I shall simply use the *meta-grammatical terms*, such as the term that occurs on the left hand side of (1), to talk *about* grammatical terms, and leave aside further issues of the nature of those terms and their relation to the expressions of the language.

2.2. Standard compositionality. Semantic functions map grammatical terms on meanings. I shall use 'μ', sometimes with subscripts or superscripts, for semantic functions. In this context, I shall also leave aside the question exactly which entities serve as meanings, whether they are extensions, intensions, structures, Fregean senses,

[8] Note that in Hodges' setup (but not here), the atomic expressions are also grammatical terms.

[9] In this sentence the grammatical term is *used*, and refers to an expression of L, while in the preceding sentence it is mentioned.

sets of proofs, or something else. We assume a set M of meanings. So, where GTA_L is the grammatical term algebra for L, $\mu : GT_L \longrightarrow M$, is a semantic function.

On the other hand, meanings are not meanings intrinsically, but only from the point of view of being the value of a semantic function. So the concepts of a *meaning* and of a *semantic function* must be taken as interdefinable and co-basic. I shall use 'm', 'm_1', 'm_2',...as parameters for meanings. At present, nothing need be assumed about the inner structure of M, the domain of meanings. In §3, where the idea of recursive semantics will be developed, an algebraic structure will have to be assumed.

There is a question whether a semantic function should be seen as *partial* or *total* on the set of grammatical terms. It has been held that *category mistakes* in the sense of Ryle give rise to expressions that are strictly meaningless, and Chomsky's famous example 'Colorless green ideas sleep furiously' has been held to be a perfectly grammatical but nonsensical sentence. I find such examples entirely unconvincing: usually falsity is a better alternative (cf. Magidor, 2009). There is also a technical reason, in that accommodating partiality computationally would involve great complexity (both in the technical and in the nontechnical sense). Since partiality isn't mandatory, technical considerations tell against allowing it.

To set out a compositional semantics, we will need certain functions that map meanings on meanings:

DEFINITION 2. *A meaning operation is a function* $r : M^n \longrightarrow M$, *for some positive natural number* n.

Meaning operations may be partial, but will be assumed to be defined in all relevant applications. I shall use 'r', with numerical and syntactic operator subscripts as parameters for meaning operations.

DEFINITION 3. *Given a grammar* GTA_L *and a domain* M *of meanings, a semantic function* $\mu : GT_L \longrightarrow M$ *is standard compositional iff for each n-ary operation* $\sigma \in \Sigma_L$ *there is a meaning operation* $r_\sigma : M^n \longrightarrow M$ *such that for any grammatical terms* $t_1, \ldots, t_n$ *for which* σ *is defined,* $\mu(\sigma(t_1, \ldots, t_n)) = r_\sigma(\mu(t_1), \ldots, \mu(t_n))$.

2.3. General compositionality. Standard compositionality can be generalized by making the meaning of a subterm depend on its linguistic context.[10] Such a generalization is of interest, since it increases semantic power (some semantic functions, e.g. for quotation, are not standard compositional but are general compositional) without increasing computational complexity. The latter aspect makes it obligatory in the present context, since the conclusion that minimally complex semantics is guaranteed to be compositional, does hold for general compositionality, but—precisely because of that—not for standard compositionality.

There are two equivalent ways of implementing this generalization: by adding an argument for linguistic context to a single semantic function, and by replacing a single semantic function by a *set* of functions. The second does not require a formal definition of linguistic context, and I shall employ it here. The idea is that a semantic evaluation

[10] Semantics which make use of this possibility have been called "switcher semantics." Switcher semantics has been employed giving a semantics for names in modal contexts (Glüer & Pagin, 2006, 2008), general terms in modal contexts (Glüer & Pagin, 2012), quotation (Pagin & Westerståhl, 2010c), and belief sentences (Pagin, 2019a). The framework is presented the latter two works and in Pagin & Westerståhl (2010a, 2010b).

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

always starts with the same, designated, function, for unembedded terms, but may shift to other functions depending on the syntactic operation and the argument position. First we need a definition of a *semantic system*:

DEFINITION 4. *Given a grammar GTA_L and a domain M of meanings, a semantic system S is a triple (S', μ_S, Ψ_S) where S' is a finite set of semantic functions $\mu_j : GT_L \longrightarrow M$, μ_S a designated member of S, and $\Psi_S : S' \times \Sigma_L \times N \longrightarrow S'$ a (partial) selection function that maps any triple of a function $\mu_j \in S'$, an n-ary operation $\sigma \in \Sigma_L$ and an argument position i of σ, $1 \leq i \leq n$, on a function $\mu_k \in S'$. For any term t, $S(t) = \mu_S(t)$.*

A system S is then said to be *general compositional* iff the semantic functions in S' can be defined by mutual recursion over syntax with composition equations analogous to those in standard compositionality:

DEFINITION 5. *Given a grammar GTA_L and a domain M of meanings, a semantic system S is general compositional iff for each n-ary operation $\sigma \in \Sigma_L$ and each member $\mu_i \in S'$ there is a meaning operation $r_{\sigma,i} : M^n \longrightarrow M$ such that for any grammatical terms $t_1, \ldots, t_n$ for which σ is defined,*

$$\mu_i(\sigma(t_1, \ldots, t_n)) = r_{\sigma,i}(\mu_{j_1}(t_1), \ldots, \mu_{j_n}(t_n)),$$

where $\mu_{j_k} = \Psi_S(\mu_i, \sigma, k)$, $1 \leq k \leq n$.

It is easy to see that in the particular case where $S' = \{\mu\}$ (for some μ), we get standard compositionality as a special case. In that case, of course, $\Psi_S(\mu, \sigma, k) = \mu$, for any σ and k.

§3. Computability and recursive semantics. Computational semantics is a part of computational linguistics where formal or model theoretic semantics is implemented algorithmically, and also combined with other computational linguistic techniques, e.g. concerning parsing. Ultimately, its goal is to provide algorithms for computing the meanings of natural language text fragments (cf. e.g. Bunt & Muskens, 1999).

The practice of computational semantics does not directly tell us what it is for meaning to be computable. In formal semantics, meanings are represented in some meta-language (ML) and rules are provided for deriving canonical meaning representations from representations that contain reference to OL expressions or grammatical terms. These rules can be more or less algorithmic. If they are fully algorithmic, the semantics is clearly computable. But in order to characterize the concept of a *computable semantic function* more generally we need a more systematic approach.

A function f on the natural numbers is *Turing computable* iff there is a Turing machine that for any representation of an argument x as input to f halts with a representation of the value $f(x)$ as output. The inputs and outputs are not numbers but syntactic representations of numbers, according to some encoding into the language of the machine. By analogy, we would say that a semantic function μ on a set of terms GT_L is Turing computable iff there is a Turing machine that for any term $t \in GT_L$ as input terminates with $\mu(t)$ as output. In this case, the inputs can be the terms themselves, but the outputs must be *canonical representations* of meanings in some chosen formal ML.

There are two crucial nontechnical differences between the machine computations in these two cases. First, in the number-theoretic case, it does not matter that the machine

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

operates on syntactic representations of numbers rather than the numbers themselves, since the relevant properties of numbers are shared by their canonical representations. The semantic case is different, since the relation between the meta-linguistic canonical meaning representation and the meaning it represents is of the same kind as the relation between the OL expressions and their meanings, the relation that we try to model by means of a ML and an algorithm. This fact raises some questions about the significance of the model, but I think the most relevant questions can be answered.

Second, while machine computation of number functions is not intended as a model of human computation, in the semantic case we do want a reasonable model of human interpretation. Turing computation isn't. According to the *Church-Turing Thesis*, any function on the natural numbers that can be computed by a human or a machine following a finite set of explicit instructions is recursive (cf. Copeland, 2008). As was established by Alonzo Church, S.C. Kleene, and Alan Turing in 1936, the concepts of being *recursive*, *lambda-definable* and *computable by a Turing machine*, are coextensive, given the domain of natural numbers. By analogy, we can hope to model human interpretation by means *recursive functions* on a syntactic domain.

However, while the concept of Turing computation is directly applicable to any domain where computations are substitution operations on strings, the concept of a recursive function is only defined for natural numbers, and so the equation of computability with recursiveness is not directly applicable. It is nevertheless indirectly applicable. As shown originally by Gödel, we can effectively code syntax in arithmetic. The semantic functions we are interested in will be represented by functions from OL terms to ML expressions; since the ML will be unambiguous, we don't need the term-expression distinction there. Both the OL and the ML can therefore be arithmetically coded, and functions from OL to ML thereby coded as arithmetical functions. Hence, the concept of a recursive function is indirectly applicable via the coding. We will define a concept of recursive ^{s} semantic function and show that the encoding image of a recursive ^{s} function is indeed recursive (Proposition 1).

To proceed, we shall first need to assume that the domain M of meanings is generated by an algebra.

DEFINITION 6. *A meaning algebra* MA_M *for a set of meanings* M *is a triple* (M, B_M, R_M), *where* B_M *is a finite set of basic meanings, and* R_M *is a finite set of elementary meaning operations.* M *is the closure of* B_M *under* R_M. *We also require that for any* $\delta, \delta' \in R_M$ *and any* $x_1, \ldots, x_n, y_1, \ldots, y_m$,

i) $\delta(x_1, \ldots, x_n) \notin B_M$.
ii) *if* $\delta(x_1, \ldots, x_n) = \delta'(y_1, \ldots, y_m)$, *then* $\delta = \delta'$, $m = n$, $x_i = y_i, 1 \leq i \leq n$.

In virtue of conditions i) and ii), there is a function $|\cdot|_M : M \longrightarrow \mathbb{N}$, where $|x|_M$ is the number of "occurrences" of elements in $B_M \cup R_M$ in x:

$(|\cdot|_M)$ i) $|x|_M = 1$, if $x \in B_M$.
ii) if $x = \delta(x_1, \ldots, x_n)$, then $|x|_M = 1 + |x_1|_M + \cdots + |x_n|_M$.

We can easily verify that $|\cdot|_M$ is well-defined.

The elementary meaning operations correspond to the successor operation for the natural numbers: they are methods of elementarily forming more complex meanings from simples meanings, such as *wooden chair* from *wooden* and *chair* by means some operation $r \in R_M$ that corresponds to an attribute-noun construction. The computability of elementary meaning operations is ensured formally: where $\ulcorner r \urcorner$ is

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

a canonical representation of r, and $\ulcorner m_i \urcorner$ of m_i, $1 \leq i \leq n$, the canonical representation of $r(m_1, \ldots, m_n)$ is simply $\ulcorner r(m_1, \ldots, m_n) \urcorner$.

We next form a two-sorted *semantic algebra* from a grammatical term algebra and a meaning algebra, together with a set of semantic functions:

DEFINITION 7. *A semantic algebra SA_{LM} for a language L and a domain of meanings M is a triple*

$$(\langle GT_L, M\rangle, \langle AT_L, B_M\rangle, \langle \Sigma_L, R_M\rangle),$$

where (GT_L, AT_L, Σ_L) is a grammatical term algebra for L and (M, B_M, R_M) is meaning algebra for M.

We shall now define the concept of a *recursive*^{s} function, in analogy with the standard definition of a recursive function on the natural numbers.[11] It will, in full analogy, be defined by three kinds of basic function and three kinds of operation for inductive definitions of new functions. The basic kinds are: successor kind operations (syntactic operations and elementary meaning operations), constant functions (one for each atomic element), and projection functions. The three kinds of inductive operations are: function composition, primitive recursion, and minimization.

Minimization requires a *computable total ordering* $\prec$ of $GT_L \cup M$.

DEFINITION 8. *We define a total ordering $\prec$ of the domain of $GT_L \cup M$. We assume given a total ordering $\prec_A$ of the domain $AT_L \cup B_M$ and a total ordering $\prec_O$ of the domain $\Sigma_L \cup R_M$. Since these domains are finite, the orderings are arbitrary, up to a point: we do require that if the arity of δ' is higher than the arity of δ, then $\delta \prec_O \delta'$.* ***We denote the least element in the $\prec_A$ ordering by '@'.***

We also define the size $|x|$ of an element $x \in GT_L \cup M$ as the number of occurrences of elements of $AT_L \cup B \cup \Sigma_L \cup R$ in x (in the sense of $|\cdot|_M$ if $x \in M$). Then

$$x \prec y \text{ iff } \begin{cases} x, y \in GT_L \cup M, \text{ and } x \prec_A y \\ |x| < |y| \\ |x| = |y|,\ x = \delta(x_1, \ldots, x_n), y = \delta'(y_1, \ldots, y_m), \text{ and } \delta \prec_O \delta' \\ |x| = |y|, x = \delta(x_1, \ldots, x_n), y = \delta(y_1, \ldots, y_n), \text{ and } x_1 \prec y_1 \\ \quad \text{or there is } k, 1 < k \leq n, \text{ s.t. } x_k \prec y_k \text{ and for all } i,\ 1 \leq i < k, x_i \sim y_i, \end{cases}$$

where $x \sim y$ iff $x, y \in GT_L \cup M$, and $x \not\prec y, y \not\prec x$.

It can be verified by induction on size that $\prec$ is total, i.e. that if $x \sim y$, then $x = y$. Moreover, if the meaning operations in R_M are computable, then $\prec$ is computable: each element in R_M is 1 – 1, and each element in M — B_M is the value of some application of some member of R_M. Hence, for each element $x \in M$, we can effectively determine that it is in B_M or that for some $\delta \in R_M$ and some $x_1, \ldots, x_n \in M$, $x = \delta(x_1, \ldots, x_n)$. Hence, for any $x, y \in M$, we can effectively determine whether $x \prec y$, $y \prec x$ or $x = y$ by applying the clauses of the definition. We shall call any such computable total ordering a *semantic algebra* ordering.

[11] This general concept of recursive^s semantics is more general than what will be needed for the complexity arguments. It is defined here to show that the more general concept is well-defined, and that it sustains Propositions 1 and 2.

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

We can now define the concept of a *recursive* $^{s}_{\prec}$ function, i.e. a semantic recursive function relative to a total ordering $\prec$. Let '$\vec{x}$' be short for '$x_1, \ldots, x_n$'.

DEFINITION 9. *With respect to a semantic algebra SA_{LM}, and a semantic algebra ordering $\prec$ on $GT_L \cup M$, a (partial) function h is recursive $^{s}_{\prec}$ iff h satisfies one of i)–vi):*

i) *h is a basic constant function such that $h(x) = y$, and $y \in AT_L \cup B_M$.*
ii) *$h \in \Sigma_L \cup R_M$.*
iii) *h is a projection function π^n_i, where $i \leq n$, is a function of n arguments such that for any arguments $x_1, \ldots, x_n$, $\pi^n_i(x_1, \ldots, x_n) = x_i$.*
iv) *h is defined by function composition*

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$$

and $f, g_1, \ldots, g_n$ are recursive $^{s}_{\prec}$.
v) *h is defined by primitive recursion ,*

$$h(\vec{x}, y) = \begin{cases} f_y(\vec{x}) & \text{if } y \in AT_L \cup B_M \\ g_\delta(h(\vec{x}, y_1), \ldots, h(\vec{x}, y_k), \vec{x}, y_1, \ldots, y_k) & \text{if } y = \delta(y_1, \ldots, y_k), \end{cases}$$

where f_y (for $y \in AT_L \cup B_M$) and g_δ (for each relevant $\delta \in \Sigma_L \cup R_M$) are recursive $^{s}_{\prec}$, and either $y \in GT_L$ and $\delta \in \Sigma_L$, or $y \in M$ and $\delta \in R_M$.
vi) *$h = Mn_{\prec}(f)$ is defined by minimization, where f is recursive$^{s}_{\prec}$. Then $Mn_{\prec}(f)(x_1, \ldots, x_n) = x$ iff $f(x_1, \ldots, x_n, x) = @$, $f(x_1, \ldots, x_n, x')$ is defined for all $x' \prec x$, and for no $x' \prec x$ does it hold that $f(x_1, \ldots, x_n, x') = @$.*

This definition largely parallels the standard definition of a recursive function on the natural numbers (cf. Boolos, Jeffrey, & Burgess, 2002). Projection and function composition are the same. Instead of a single successor function, each element in $\Sigma_L \cup R_M$ has the role of a successor function. Instead of a single basic constant function giving 0 as value, there is a constant function for each element in $AT_L \cup B_m$. Analogously for primitive recursion: instead of a single basic step, there is one basic step for each element of $AT_L \cup B_m$. Also, the recursion must distinguish between the case where the successor operation belongs to Σ_l and the case where it belongs to R_M. Finally, for minimization we need a total order $\prec$, but this has no natural definition within the algebra itself, and must be imposed from outside. If we were to use arithmetic coding, $\prec$ would simply be natural ordering of the natural numbers.

With respect to a semantic algebra SA_{LM}, a recursive$^{s}_{\prec}$ *semantic* function is a recursive$^{s}_{\prec}$ function from GT_L to M.

We can quickly convince ourselves that recursive$^{s}_{\prec}$ functions are computable. The successor operations are computable, constant functions with specified values are computable. Projection functions are computable. Function composition, primitive recursion and minimization preserve computability (since the semantic ordering relation $\prec$ is required to be computable).

Furthermore, we can show that recursive $^{s}_{\prec}$ functions can be represented by recursive number theoretic functions. The proof of the following fact is given in Appendix A.

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

PROPOSITION 1. *Representation theorem*

Given a semantic algebra SA_{LM}, *there is a semantic algebra ordering* $\prec$ *of* $GT_L \cup M$ *and a Gödel numbering* $\ulcorner \cdot \urcorner$ *from* $GT_L \cup M$ *to* $\mathbb{N}$, *to the effect that for any recursive*$^s_\prec$ *function* $f : (GT_L \cup M)^n \longrightarrow GT_L \cup M$ *there is a function* $\overline{f} : \mathbb{N}^n \longrightarrow \mathbb{N}$ *such that*

a) *for any* $x_1, \ldots, x_n \in GT_L \cup M$, $\ulcorner f(x_1, \ldots, x_n) \urcorner = \overline{f}(\ulcorner x_1 \urcorner, \ldots, \ulcorner x_n \urcorner)$,
b) $\overline{f}$ *is recursive.*

Hence, the concept of a recursive$^s_\prec$ function is well-defined and analogous to the concept of a recursive function. However, despite the fact that use of Minimization is essential for *computability in principle*, there are three reasons to restrict our attention to semantic functions whose definitions do *not* require Minimization, i.e. to the semantic counterpart of *primitive recursive functions*. These are the functions definable by clauses i)–v) of Definition 9.

The first reason is that Minimization requires a total order of the objects of the algebra, and there is no total order in a semantic algebra that is defined from the basic operators themselves. At least there are two separate subdomains, GT_L and M, and on top, neither need have just one unary operation. So a total order apparently needs to be imposed from outside.[12] This breaks the analogy with arithmetic.

The second reason is that it is very implausible that Minimization would ever be an operation involved in human interpretation. So to the extent that this is what we want to model, there is reason not to include Minimization.

The third reason is that from the point of view of computational complexity, the primitive recursive functions suffice. In arithmetic, all *tractable* functions (see §4) and many more are primitive recursive (cf. Immerman, 2008). And, as we will see in §7, the results from considering only primitive recursive$^s_\prec$ functions are strong enough.[13] From now on, unless otherwise noted, by 'recursive^s semantic function', and 'recursive^s operation', I shall mean a semantic function or an operation definable by clauses i)–v) of Definition 9.

Then we can define:

DEFINITION 10. *Given a semantic algebra* SA_{LM}, *a semantic function* $\mu : GT_L \longrightarrow M$ *is recursive*^{s} *iff* μ *is definable by clauses i)–v) of Definition* 9.

From now on, I will drop the superscript, except for cases where disambiguation is required.

Now, it is clear that a compositional semantic function need not be recursive. For the semantic function μ is recursive just in case the meaning operations are recursive, but it is not required in Definition 3 (or in any common definition of compositionality) that meaning operations be recursive.

It is also clear that a recursive semantic function need not be compositional.[14]

Using primitive recursion in the definition of μ we will have the condition that

(REC) Let $\overline{R}_M$ be the recursive closure of R_M. Then for each n-ary operation $\sigma \in \Sigma$ there is a recursive operation $r_\sigma \in \overline{R}_M$ such that for all terms $t_1, \ldots, t_n$, if μ

[12] This remains to be proved.

[13] It is not known whether all semantically computable functions are also semantically recursive. An earlier argument for the truth of that proposition turned out to be flawed.

[14] This has been observed many times before, e.g. in Janssen (1997), by means of an example from arithmetic.

is defined for $\sigma(t_1, \ldots, t_n)$, then

$$\mu(\sigma(t_1, \ldots, t_n)) = r_\sigma(\mu(t_1), \ldots, \mu(t_n), t_1, \ldots, t_n).$$

The fact that the meaning operations may take the syntactic terms $t_1, \ldots, t_n$ *themselves* as arguments, has the effect that the compositional substitution laws need not hold. For

$$r(\mu(t_1), \mu(t_2), t_1, t_2)$$

may well differ from

$$r(\mu(t_1), \mu(t_3), t_1, t_3)$$

even if $\mu(t_2) = \mu(t_3)$. Hence, recursiveness does not entail compositionality.

Then, given that recursive semantic functions are computable, a single example of a recursive semantic function that is not compositional is enough to show a computable semantic function need not be compositional.[15] An example will be given in term rewriting format in §4. We now turn to complexity.

§4. Complexity and efficiency. In computational complexity theory, three types of measure have been studied extensively, all defined in terms of Turing machines:

The *time complexity* of a problem P (relative to a way of describing P) with respect to an algorithm A, is the maximum number of computation steps that are needed for a Turing machine that implements A to solve a problem instance of the same *size* as P (cf. Garey & Johnson, 1979, pp. 6, 26).

The *space complexity* of a problem P (relative to a way of describing P) with respect to an algorithm A, is the maximum number of distinct tape squares visited by a Turing machine that implements A to solve a problem instance of the same *size* as P (cf. Garey & Johnson, 1979, p. 170).

The *Kolmogorov complexity* of a problem P (relative to a way of describing P) is the size (relative to a linear encoding of Turing machines) of the smallest Turing machine needed to solve P (cf. Li & Vitányi, 1997, pp. 93–98).

In these contexts, a "problem" is usually a problem *type*, and what is solved in each particular case is an *instance* of that problem type. Such a problem type is e.g. *The Traveling Salesman*: a salesman is to visit a number of cities exactly once and then return home, and the task is to find a visiting order that minimizes the total distance traveled. The number of cities is the *size* of the instance.

Depending on the choice of complexity measure and on the choice of method for solving problems of a chosen type, a particular problem instance gets assigned a natural number as the complexity of that instance, numbering e.g. the squares that a chosen Turing machine has visited for computing the solution. One is interested not only and not primarily in the complexity of individual instances, but in the *maximal complexity* of instances of the same *size*: given e.g. a number of cities to visit (disregarding further information), what is the maximal number of steps needed to determine the best order? Given a problem type P and a method ψ, one is therefore interested in the *complexity*

[15] As pointed out in Werning (2005) and developed in Pagin & Westerståhl (2010c), we can have a natural recursive semantics for quotation that is not compositional. An example can be found in Potts (2007). Quotation is, however, given a semantics that is *general compositional* in Pagin & Westerståhl (2010c).

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

function $C_{P,\psi}$ from natural numbers to natural numbers which, for a given argument k gives as value the maximal complexity of P-instances of size k.

Suppose we can hold P constant. We can then simply associate each *method* ψ with a complexity function C_ψ. When each method is associated with a complexity function, we can compare methods with respect to *efficiency*: if for all k it holds that $C_\psi(k) < C_\varepsilon(k)$, then we can say that method ψ is more *efficient* than method ε. In general, the efficiency comparison is less straightforward, since one method may be more efficient than another only in the long run. The most natural way of capturing this idea is to require that from some size onwards the C values are lower for the one than for the other:

(EC) ψ is more efficient than ε iff there is an n such that

$$\forall k > n\ (C_\psi(k) < C_\varepsilon(k)).$$

We are interested in finding out the properties of those interpretation methods, i.e. semantic functions, that are *most efficient* in this sense. As we shall see below, however, this comparison is significant only for large differences.

§5. Measures of complexity. We turn to the question of selecting an appropriate complexity measure. Which of the three aforementioned types of complexity are relevant to an intuitive measure of the difficulty of the cognitive task? Time complexity appears to be most directly relevant, since considerations of time pressure motivated looking at complexity in the first place.[16] The general idea of time complexity is that we count the number of steps that have to be taken for completing a process. It is plausible to assume that by and large, an increase of the number of steps required in the formal computation corresponds to an increase of the real time needed for human processing. I shall assume so. Hence, we will in the first place want to minimize time complexity of individual tasks, and therefore to maximize efficiency of semantic functions with respect to time complexity.

The next questions to answer are: What is the character of the relevant problem type? How do we measure the size of an instance? What constitutes an individual step of the computation process?

It would be very natural at first glance to take efficiency maximizing to consist in finding the most efficient semantic function for a given language L and a given domain M of meanings, where L is understood purely syntactically. That is, measure the size of the problem instance as the size of the term, and look for the longest computation needed for a term of that size. But although this question is important, computation

[16] Henry (2005) appeals to Kolmogorov complexity in supporting compositionality from a learning perspective. A machine is taken to generate string-meaning pairs, and complexity is low if the length of the description of the machine is small in relation to the size of the data it outputs. Compositionality offers low complexity in this sense. And if the language learner is equipped with a compositionality hypothesis, the learning of a language with compositional semantics will be much faster than if meanings are paired with strings in a more random way. Hence, Brighton concludes, it makes sense from an evolutionary perspective to believe that natural languages are compositional.

This is no doubt true, but any systematic semantics would have the same advantage over random pairing, and so the result does not throw much light on compositionality in particular. Cf. Pagin (2013).

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

length relative to input size cannot be the whole story. For with some choices of L and M, semantic interpretation becomes *intractable*, whatever the semantic function. To see this, consider the following example of **Lisa** (cf. §8.3.1 for the term rewriting system).

We have a language L consisting of α and the successor operator σ as basic elements. So L consists of the terms $\alpha, \sigma(\alpha), \sigma(\sigma(\alpha))$, etc. Then we have a conceptual domain consisting of the object l (Lisa) and the two conceptual functions f (father) and m (mother). So M consists of $l, m(l), f(l), m(m(l)), m(f(l)), f(m(l)), f(f(l))$, etc. Let the *size* of an object of either domain consist in its number of basic elements (basic object plus number of function/operator occurrences), and let the *size n* of a domain be the number of elements of at most size n that it contains. The *growth rate* of the domain is a rate of increase of the function $g(n)$ that maps a number n on the number of elements of the domain with size n or lower. In these terms, we have a conceptual domain with a growth rate of $g(n+1) = 2g(n) + 1$ and a syntactic domain with a growth rate of $g(n+1) = g(n) + 1$.

In order that the semantic function μ for each element m in M up to size n maps a term t on m, $2^{n+1} - 1$ distinct terms will be needed, since there are that many elements of M of size n or lower. Therefore, there will be at least one element m_i and one term t_j of L such that $\mu(t_j) = m_i$ and the size of t_j is $2^{n+1} - 1$ or greater. Assuming that exactly one computation step is needed for processing each element of a term t, at least $2^{n+1} - 1$ steps are needed to compute t_j.[17] Hence, the maximal number of steps needed to process a term referring to an object of a certain size grows *exponentially* with the size of the object, regardless of the choice of semantic function. In terms of computational complexity theory, using L for referring to M in the **Lisa** example is a strictly *intractable* task. When tasks are considered tractable, the increase in time complexity (number of steps needed) is at most a polynomial function (known as the *Cobham-Edmonds* thesis).

More precisely, we get the intractability result if we regard the *problem type* as the type of understanding the expression of a concept, for then the size of a problem instance is the size of that concept. If by contrast we regard the problem type as that of processing a *term*, then the size of the problem instance is the size of the term. The time complexity function, under the assumption above, is then simply x: as many steps are needed as there are elements in the term. From that point of view, the semantic function appears very efficient. For the same conceptual domain M we could have a more appropriate language L' and a semantic function μ' mapping L' on M that would require, say, terms that are twice as large as the concepts they are mapped on. With the same assumption of one step per element of the term, μ would be as efficient as μ' if the problem type is that of processing terms, but exponentially less efficient if the problem type is that of arriving at contents. In this particular example, the latter is clearly what is cognitively more relevant: $L + \mu$ form an intractably cumbersome combination for talking about or expressing M, while the alternative $L' + \mu'$ would be manageable.

As a conclusion from this, we need to consider the task of *interpreting expressions of content* as a problem type, and hence to treat as a measure of the size of an instance of a problem simply the size of the content to be expressed. That is the invariant factor in

[17] This assumption is made for ease of exposition. It does not matter much. It will in any case hold that there is a finite number k such that *in the long run*, at most k elements can be computed in each step. Then there is still exponential growth.

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

the comparison between methods in the example above.[18] We have to take account not only of the efficiency of the mapping from code to concepts, but also of the efficiency of the encoding itself, i.e. the size of the code.[19]

We shall return below to the question of how to measure sizes of contents. But we must first note that just as the relation of computation length to the size of the input term can be misleading with respect to real complexity, so can the relation to the output. For it may be that a computation grows exponentially in relation to the size of input, but that the complexity measure in relation to output size is nonetheless *low* for the simple reason that the output itself grows exponentially. We will in fact see an example of this in §8 (the **Godzilla** example). The upshot is therefore that we need to care *both* about the *input complexity* C_ε^i of a method ε, i.e. the length of a computation in relation to the size of the input, and the *output complexity* C_ε^o, the length of a computation in relation to the size of the output. Both need to be reasonable.

But how do we measure the size of output, i.e. contents? Does it make sense to say that one concept or one proposition is larger than another? There is no immediate way of making a relevant sense of that idea, but it does not matter so much. Computations can anyway not be defined over contents, only over symbols. What we can and must do, then, is to measure the size of *representations* of content. We shall need a formal language where we give *canonical* representations of conceptual contents. With such a formal, unambiguous language of canonical representations, we can again count the number of symbols in its expressions for determining the relevant size of contents represented.

The final question concerns the nature of the computation steps that are to be counted. As mentioned above, standard time complexity takes the number of operations of Turing machines as the measure. If that were the choice, we would have to settle for some particular *kind* of Turing machine, whether a standard single-tape machine with a tape that is infinite in both directions, or something else. There is no uniquely right choice, and no absolute measure. Turing machine operations will

[18] A further reason not to use only the term size as the size of the problem instance is that we can make terms arbitrarily much larger by throwing in junk constituents that are not needed for the semantics and therefore do not add to computation complexity. With a lot of junk in the terms, a semantic function can appear to be more efficient, which is again counterintuitive.

[19] This aspect of the issue shows the similarity with questions of efficient encoding handled in Mathematical Information Theory, as originated in Shannon (1949). There are also important differences, however. An encoding E is efficient in the information theoretic sense if the average rate of information sent over an information channel and encoded by E is high. In that context, a signal conveys more information if the fact that it reports is less probable. States of affairs that are highly probable will in the long run occur more often, and should be reported by means of shorter codes. So the efficiency of an encoding depends on the matching between the distribution of lengths of codes and the distribution of probabilities, over the same possible states of affairs.

In the present case, the questions of truth or falsity of sentences used or the probabilities of facts reported on, do not arise. We are only concerned with the expressive power and the efficiency of the interpretation. In the information theoretic case, questions of efficient encoding arise even if there is only a small finite number of signal types (sentences) used over and over. In the present context, having only a finite number of sentences would reduce the interpretation problem to triviality, since then the meaning of all sentences could be given by a finite list. This would reduce the total number of processing steps needed for any sentence to exactly 1.

involve steps needed in order to find the relevant information (on other tape squares) and moving symbols in order to make room for others, etc., and how many such operations are needed will depend on the choice of machine. Therefore, these operations are to some extent arbitrary, and to that extent less essential to the complexity measure.

There is a natural alternative, which is to employ the *equation system* used for defining a function also as a method for computing the function. Take as a simple example, Donald Davidson's **Annette** (Davidson, 1967, pp. 17–18):

$$\begin{aligned} &\text{i)}\quad \text{Ref}(\text{`Annette'}) = \text{Annette}. \\ &\text{ii)}\quad \text{Ref}(\text{`the father of'} \frown t) = \text{the father of Ref}(t). \end{aligned} \tag{2}$$

This simple definition provides a method for deriving the interpretation of 'the father of the father of the father of Annette' in four steps of substitution. Let 'F' be the OL father operator and '**F**' its analogue in the ML, and let 'a' be the OL name of Annette. Then we have in four steps with the semantic function μ_a:

$$\begin{aligned} &\mu_a(F(F(F(a)))) \\ &= \mathbf{F}(\mu_a(F(F(a)))) \\ &= \mathbf{F}(\mathbf{F}(\mu_a(F(a)))) \\ &= \mathbf{F}(\mathbf{F}(\mathbf{F}(\mu_a(a)))) \\ &= \mathbf{F}(\mathbf{F}(\mathbf{F}(\text{Annette}))), \end{aligned} \tag{3}$$

where (what corresponds to) the second clause of (2) is applied three times and the first clause once.

Each derivation step in (3) is a substitution step. Each substitution is performed in accordance with equations in (2). These equations are applied only for substitution from left to right: an instance of the left-hand side is replaced by the corresponding instance of the right-hand side. This makes the system into a so-called *term rewriting system*.

Term rewriting systems are sets of *rewrite rules*. Rewrite rules apply to *terms* and license formal *substitutions* of/in those terms. Rewrite rules can contain variables, in which case an *instance* of the left-hand-side is allowed to be transformed to the corresponding instance of the right-hand-side. Clause ii) of (2) can be regarded as such a rule, with the variable t occurring once on each side. Relative to some rewriting system R, when no rule of R applies to a term u, u is said to be in *normal form*. The little derivation in (3) transforms the initial term '$\mu_a(F(F(F(a))))$' to its normal form '$\mathbf{F}(\mathbf{F}(\mathbf{F}(\text{Annette})))$' in four steps.

Transforming terms to normal form by means of a sequence of rewrite rule applications is a completely general form of computation. It has been shown that any both-way infinite one-tape Turing machine can be simulated by a term rewriting system such that each rule of the rewriting system corresponds to a machine transition and each machine transition is represented by at least one rewrite rule (cf. Baader & Nipkow, 1998, pp. 94–97). In virtue of this relation it is not only very convenient but also well motivated to use the count of rewrite rule applications as a measure of time complexity.

Then, for each non-normal rewriting term s, we consider the shortest derivation by which s is normalized. Only normal terms correspond to full interpretation, i.e. to our representations of the world; other terms only have a role in deriving the normal terms. Let *input terms* of the rewriting system be terms of the form '$\mu(t)$', with t a syntactic

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

term. For a *normal* term s we consider the *shortest* derivation by which some input term $\mu(t)$ is normalized to s.[20] Let that be the *term complexity* $Ct_R(s)$ of s relative to R. The input time complexity $C_R^i(k)$ for the size k relative to the system R is then the *maximal* $Ct_R(s)$ such that s has size k. We have the corresponding definition of the *output* time complexity $C_R^o(k)$ in relation to the size of the output expression in normal form.

With this much of background, I turn to characterizing time complexity of rewrite systems in relation to some crucial properties of those systems.

§6. Term rewriting.

6.1. Term rewriting and u systems. A *term rewrite* system (TRS) Φ is a pair (Σ_Φ, R_Φ) of a *signature* Σ_Φ and *rule-system* R_Φ over the signature.[21] The signature consists of a set of *operator* symbols of different arities, including the null-arity (for constants). We shall here require that Σ_Φ is finite. $\mathcal{T}_\Phi$ is the set of *terms* over Σ_Φ.

The rule-system provides transformations, or *reductions*, from terms to terms in $\mathcal{T}_\Phi$. The rules are stated by means *rewrite variables*, normally from a denumerable set $\mathcal{V}_\Phi = \{x_1, x_2, ...\}$. Rules are then of the form

$$F(\overrightarrow{x}) \rightarrow G(\overrightarrow{y})$$

(where the arrows over the variables indicate that it is a sequence of variables). The main arrow indicates that any uniform *substitution instance* of the lhs (left-hand-side) *reduces* to the corresponding substitution instance of the rhs (right-hand-side). The lhs and rhs of rule formulations are *patterns* or *schemes* formed from $\mathcal{T}_\Phi \cup \mathcal{V}_\Phi$.

A rule example would be

$$h(x_1)bx_2 \rightarrow g(x_1, c)bd,$$

where 'b', 'c' and 'd' are constants. Assuming t_1, t_2 are terms in $\mathcal{T}_\Phi$, application of the rule reduces '$h(t_1)bt_2$' to '$g(t_1, c)bd$'. Since reductions can take place in contexts of larger terms, the same rule reduces '$j(h(t_1)bt_2)$' to '$j(g(t_1, c)bd)$'.

A *derivation* is a sequence of rule applications in a rule-system, where the result (called "contractum") of each application except the last provides the input (called "redex") to the next. A term that cannot be reduced further is said to be in *normal form*. A rewrite system Φ is said to *terminate* iff every derivation in R_Φ eventually leads to a term in normal form. R is said to be *confluent* iff it holds for any terms t_1, t_2, t_3 such that both t_2 and t_3 can be derived from t_1, that there is a term t_4 such that t_4 can be derived from both t_2 and t_3. R is *convergent* iff R both terminates and is confluent. Not all term rewriting systems terminate and not all are confluent, and neither property is in general decidable. However, the systems we will be concerned with, involve substitutions of a very restricted kind, and they are convergent, as we shall see.

For our purposes we will need two generalizations of standard TRSs. The first is that of *many-sorted* systems (cf. Terese, 2003, pp. 254–259). In a many-sorted TRS, the set of terms is partitioned into subsets, where each such subset is the set of terms of a distinct *sort*. Rewrite variables are also typed for sorts, so that the possible instances

[20] There need not be any longest derivation, since it is possible that there is no upper bound to the size of terms that reduce to the same normal form.

[21] For excellent introductions to term rewriting, see Baader & Nipkow (1998) and Terese (2003).

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

are terms of only one sort. A complex term t of a sort s_i can have subterms of different sorts $s_{j_1}, \ldots, s_{j_n}$. Argument places for operators are then associated with *sort restrictions*. A complex term all of whose subterms satisfy the sort restrictions is called a *well-sorted* term. This of course is the counterpart to grammaticality.

The first reason why we need many-sorted systems is that they will handle both an OL and a ML. We will then have an OL sort and an ML sort. In case the ML is an extension of the OL, this need for a sort distinction vanishes, but in the general case, the ML does not overlap at all with the OL. In particular, in the normal case, the OL contains structurally ambiguous expressions. Since semantics only assigns meaning to disambiguated expressions, we will need a special set of unambiguous *grammatical terms* that are the arguments to the semantic functions. In the present context, we will not be concerned with the expressions of the OL at all, only with the grammatical terms. The ML, on the other hand, is *required* not to contain structural ambiguity at all. It shall provide canonical meaning representations, and hence we will require that distinct ML expressions represent distinct meanings. No grammatical terms for the ML will therefore be needed. For this reason, where there is no risk of confusion, I shall refer to ML terms simply as "expressions."

There will be a set of rewrite variables $v_1, v_2, \ldots$ of the sort OL, i.e. variables that take OL grammatical terms as substituends. But just having one sort for OL terms risks not respecting the syntactic restrictions of the object-language. We will need to instantiate schemes like

$$\mu(\sigma(v_1, \ldots, v_n)),$$

where μ is a semantic function symbol, σ is an OL syntactic operator, and $v_1, \ldots, v_n$ are rewrite variables. The argument of the instance must be a grammatical term $\sigma(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ themselves are grammatical terms. But no operator σ is defined for all sequences $t_1, \ldots, t_n$ of grammatical terms. If σ is the NP-VP operator, then there are two arguments, where the first must be an NP and the second a VP. So we need to impose restrictions on instantiations.

One method for doing this is to introduce further subcategorizing of the OL sort into sorts that correspond to grammatical categories, like NP. I shall here use the other method (which is anyway needed for further reasons discussed below): generalizing TRSs to *conditional term rewrite systems* (CTRSs). A CTRS Φ is again a pair (Σ_Φ, R_ϕ) of a signature and a set of rules, except only that the rules in general will be *conditional*. A conditional rule r is an ordinary rule together with conditions for its application[22] :

$$t \rightarrow s \qquad \Leftarrow \qquad C_1, \ldots, C_n.$$

We can apply this method to implement the grammaticality restriction, by simply adding the condition that the complex term $\sigma(t_1, \ldots, t_n)$ is grammatical. I shall choose this method for reasons of simplicity.

There would be a second reason why both many-sorted and conditional rewriting would be needed for some systems. This reason would concern the existence of variable binding operators in the OL, and the need that it induces for varying semantic

[22] Cf. Terese (2003), pp. 80–85.

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

assignments to free variables. However, variable binding in the OL introduces a number of complications, and cannot be treated in this paper.[23]

The systems we shall focus on are transformed into rewrite systems from algebraic specifications of semantics for some OL. They will be called *μ-systems*.

DEFINITION 11. *A μ-system* Φ *is a pair* (Σ_Φ, R_Φ) *of a signature and a set of rules. Every μ signature* Σ *is a pair* (Σ_o, Σ_m) *of an object-language signature* Σ_o *and a meta-language signature* Σ_m.

Σ_o *is pair* (A_o, Σ'_o) *of a set of atomic OL terms* A_o *and a set* Σ'_0 *of n-place syntactic operators* $\sigma \in \Sigma'_o$, *for variable n. The operators in* Σ'_o *are in general partial.*[24] *The set of OL terms* $\mathcal{T}_o$ *is the closure of* $A_o \cup V_o$ *under* Σ'_o. *Thus* $(\mathcal{T}_o, A_o, \Sigma'_o)$ *is a syntactic algebra, the grammatical term algebra of OL.* $\mathcal{T}_o$ *is the set of OL grammatical terms.*

Σ_m *is a structure* $(A_m, \Sigma'_m, S, F, G)$ *of a nonempty set* A_m *of atomic ML expressions, a nonempty set* Σ'_m *of ML syntactic operators, a nonempty set S of ML elementary semantic function symbols, and two (possibly empty) sets F and G of additional meta-linguistic recursive function symbols. They are needed for so-called indirect systems. Members of F perform recursion over OL terms and members of G over ML expressions.*

$\mathcal{C}_m$ *is the closure of* A_m *under* Σ'_m. *It is the set of canonical expressions of ML, which are canonical representations of meanings. It does not contain any μ operator or F or G functor.*

S is the set of semantic vocabulary of Φ. *It contains at least one member, the semantic function symbol 'μ'. In the general case, S may be a finite set* $\{\mu_1, \ldots, \mu_n\}$ *of semantic function symbols.* $S(\mathcal{T}_o) = \{\mu_i(t) : t \in \mathcal{T}_o, \mu_i \in S\}$ *is the set of μ-terms. It is the set of symbols consisting of a semantic function symbol applied to an OL term.*

F may include additional function symbols f_1, $f_2, \ldots$ *that do not represent semantic functions strictly speaking, but are involved in the computation of semantic values. Members of F can take arguments from both* $\mathcal{T}_o$, $S(\mathcal{T}_o)$ *and* $\mathcal{C}_m$. *They are part neither of the input to* R_Φ *nor of the normal form, but play a role in recursion. Again the operators in* Σ'_m *and the function symbols in F are in general partial. The same holds for G.*

$\mathcal{T}_m$ *is the set of meta-linguistic terms of* Σ_M. $\mathcal{T}_m$ *is the closure of* $A_m \cup \mathcal{T}_o$ *under* $\Sigma'_m \cup S \cup F \cup G$. *There are two privileged subsets of* $\mathcal{T}_m$: $S(\mathcal{T}_o)$ *is the set of input terms to derivations, and* $\mathcal{C}_m$ *the set of canonical expressions, i.e. expressions in normal form, with which rewrite derivations terminate.*

In §8 we shall characterize systems of the different kinds and determine the complexity of examples. First, we shall look at the general issue of complexity and term rewriting.

§7. Term rewriting and complexity. Any terminal symbol occurring in a term is produced by means of a rule where it is explicitly used on the rhs. At most finitely many terminal symbols can occur on the rhs of any rule. Since the rule system is finite, there is a largest number w of terminal symbol occurrences that can be produced in

[23] In Pagin (2012b) I have devised a rewrite system for a first-order language, with the existential quantifier as only variable-binding operator. The requirement of matching quantifiers with variables in the rewriting process increases complexity, making it quadratic instead of linear.

[24] Here we follow the format of Hodges (2001), also used e.g. in Pagin (2003), Westerståhl (2004), and Pagin & Westerståhl (2010a), where we avoid introducing syntactic sorts, in contrast e.g. to Montague (1973), Janssen (1997), Hendriks (2001).

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

any single rule application. This is the MaxApp number of the system. It is immediate that the smallest number of application steps needed to produce a term s of size k in a μ system R is $[k/\omega]$, where ω is the MaxApp number of R and $[z]$ is the smallest whole number at least as great as z. Since there are infinitely many terms of normal form, the ratio $[k/\omega]$ will be an upper limit of efficiency in the long run. Hence, no μ rule system R can be faster than having a linear time complexity function C_R.

The real efficiency may be much lower. If rules that produce new nonterminal symbols are present, the upper limit of efficiency may be an exponential function of the size of the canonical terms. If the rule system is *direct*, every new symbol on the rhs of a rule is a terminal symbol. In virtue of property (SDCiv) of μ systems, that term cannot itself be an argument, i.e. instantiate the lhs of a rule. Only its proper subterms can. In direct μ systems, substitutions are only performed on subterms that do not contain terminal symbols. Because of this, μ systems that are direct are guaranteed to transform terms to normal form in an incremental fashion, in each step replacing nonterminal by terminal symbols, until only terminal symbols remain.

Let the MinApp of a μ system be the minimal number of terminal symbol occurrences that are produced by any single rule application. Hence, for a direct system R, MinApp $(R) \geq 1$. This means that for a direct rule system R, we can estimate the complexity function C_R as

$$[k/\omega] \leq C_R(k) \leq [k/\sigma],$$

where ω is MaxApp (R) and σ is MinApp (R). Since $\sigma \geq 1$, it follows that $C_R(k) \leq k$ if R is a direct rule system. I shall say that systems with such a complexity function are *maximally time efficient*.

Clearly, since there is no finite upper bound the value of ω, there is no highest efficiency value. Where we have a system R_μ that computes a function μ we can devise a system R'_μ that computes the same function μ at roughly twice the speed. We do this by creating more complex rules, i.e. rules that apply to larger terms. Such rules are more specialized, and hence many such rules are needed for having an equivalent system.[25] It still makes sense to speak of maximal time efficiency, for the reason that rewriting computation can be sped-up by more than any finite factor. Any speed-up of a system whose complexity function is linear is therefore itself linear. Therefore, we can say more generally that systems with a linear complexity function have *minimal complexity*.

Direct rule systems have minimal complexity.[26] What happens when we allow *indirect* rules, i.e. rules that produce nonterminal symbols that will be eliminated in the course of the composition? Term rewriting systems that implement recursive semantics will be indirect, since they produce OL terms in the ML, and these OL terms will have to be eliminated. Therefore, it seems that to the extent that indirectness of a μ system leads to an increase of computational complexity, this has the effect that recursive noncompositional semantics is less efficient than recursive semantics that is compositional as well. In §8 we shall consider kinds of indirect systems

[25] This corresponds to speed-up transformations of Turing machines. For a given Turing machine M we can e.g. devise a machine M′ that is twice as fast by letting the new one process two tape squares at a time (cf. Hartmanis & Stearns, 1965).

[26] This topic was investigated in Pagin (2012a).

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

and the complexity of examples of these kinds. We shall return to the question of compositionality and complexity in §9.

We shall see in the next section that allowing for indirect rules can lead to exponential complexity, i.e. $C(k) \geq i^k$, for some i, and even higher. This is a dramatic increase of complexity, as measured by the number of term rewriting steps, and we may wonder whether this is a reasonable measure. The question divides into two subquestions. The first concerns the relation between cognitive difficulty and abstract complexity measures, and the second concerns more specifically the relation between term rewriting complexity and standard Turing machine complexity. Neither question will be investigated here, but a few remarks will be made regarding the second question.[27]

The standard machine model for time complexity is the (multitape) Turing machine. The question whether some other computational device, like a term rewriting system, offers a significant complexity measure is known as the question whether it provides a *reasonable machine model* for time complexity, in the sense of van Emde Boas (1990). A time complexity measure by an alternative machine model R is counted as reasonable if that machine can be simulated by a Turing machine T with at most polynomial overhead:

$$\text{There is a } k \text{ such that } C_{\mathrm{T}}(n) \leq C_R(n)^k.$$

In Dal Lago & Martini (2009) it is shown that so-called *orthogonal constructor* term rewriting systems can be simulated with linear overhead by reductions into a weak type-free call-by-value Lambda Calculus. In another recent paper by the same authors (Dal Lago & Martini, 2008), it is shown that Weak Lambda Calculus can be simulated by a Turing machine with polynomial overhead ($k = 4$).

μ systems are constructor systems and orthogonal, but differ in being two-sorted, and in the use of conditions. Therefore, the results of Dal Lago and Martini cannot be directly applied, but they give reason to believe that corresponding results may be available for μ systems, and that therefore the question whether μ systems are reasonable machine models can be answered affirmatively.

§8. Direct and indirect μ systems.

8.1. SDC systems. In a *simple* μ system, S contains only one semantic function symbol μ. In a *direct* μ system, the sets F and G are empty. In a *constant* μ system all atomic terms get constant value assignments: the OL does not contain variable-binding operators. We shall here be concerned only with constant systems. We shall first outline the set of rules for *simple*, *direct*, *and constant* μ-systems, SDC systems. A set of rules R_Φ for an SDC μ system Φ has the following properties:

(SDC) *A SDC rule system R*:

i) R_Φ has a finite number of rules;
ii) For any rule r of the μ system, the set of rewrite variables on the rhs of r is a subset of the set of rewrite variables occurring on the lhs of r;

[27] For the first, see for instance van Rooij (2008).

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

iii) Rewrite variables take all and only *OL* grammatical terms as instances;
iv) Every atomic rule $r \in R_\Phi$ has the form

$$\mu(t) \rightarrow e,$$

where $t \in A_o$ and e is a simple or complex expression in $\mathcal{C}_m$, and for every $t \in A_o$ there is an atomic rule;
v) Every complex rule $r \in R_\Phi$ has the form

$$\mu(\sigma(v_1, \ldots, v_n)) \rightarrow K(\mu(v_1), \ldots, \mu(v_n)) \quad \Leftarrow \quad \mathrm{Gr}(\sigma(v_1, \ldots, v_n)),$$

where K is a simple or complex operator of Σ'_m and $\mathrm{Gr}(\sigma(v_1, \ldots, v_n))$ is the condition that the substitution instance $\sigma(t_1, \ldots, t_n)$ of the application is grammatical; and for every operator $\sigma \in \Sigma'_o$ there is a unique complex rule.

A few comments are in order. The requirement (i) that the rules are finite in number is not a general requirement on rewrite systems, but is highly motivated when we use rewrite systems as a model of the human interpretation faculty. The requirement (ii) that the set of variables in the rhs is a subset of the variables on the lhs is a general requirement on TRSs. That this condition is satisfied follows from clause (v).

From clauses (iv) and (v) we can see that for every grammatical term $t \in \mathcal{T}_o$, $\mu(t)$ is an instance of the lhs of some rule; i.e. every OL grammatical term gets processed. It is of course a further question whether every grammatical term gets fully interpreted, but we shall see that this is in fact the case. This means that the semantic function μ that is formally represented by the rule system is a *total* function. One cannot in general require that semantic functions be total, but since the syntactic operations are anyway partial, it is most convenient to count any μ meaningless term as ungrammatical as well.

The combination of clauses (iii) and (v) entails that every complex grammatical term has grammatical terms as immediate constituents, since only grammatical terms can instantiate the variables in clause (v). Given that grammatical terms are also μ meaningful, it follows that the immediate constituents of μ meaningful complex terms are themselves μ meaningful. Since Hodges (2001), this is known as the *Domain Principle*. Note also that, in virtue of clause (v), Φ represents a compositional semantics. We can establish some properties of SDC systems.

PROPOSITION 2. *SDC systems terminate.*

Proof. We can give a simple size measure of terms, as follows:

i) for $e \in \mathcal{C}_m$, $|e| = 0$,
ii) for $e = K(e_1, \ldots, e_n)$, where $e_i \in \mathcal{T}_m$, $1 \leq i \leq n$, and K is a simple or complex operator of Σ'_m, $|e| = |e_1| + \cdots + |e_n|$,
iii) for any $t \in \mathcal{T}_o$, $|\mu(t)| = |t|$,
iv) for any $t \in A_o$, $|t| = 1$,
vi) for any n-place $\sigma \in \Sigma'_o$, any terms $t_1, \ldots, t_n$, $|\sigma(t_1, \ldots, t_n)| = 1 + |t_1| + \cdots + |t_n|$.

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

It is straightforward to see that every rule application lowers the size of the ML expression by 1. That is, the redex of the application has a size one unit greater than the contractum. Since terms in $\mathcal{T}_o$ are well-founded, and no rule applies to any expression of size 0, every derivation comes to an end. □

PROPOSITION 3. *SDC systems are confluent*.

Proof. In every rule of a SDC system, the lhs has the form $\mu(v)$. No expression e in ML has an occurrence of μ within the scope of another occurrence of μ. For a complex term $\sigma(t_1, \ldots, t_n)$, the applicable rule is uniquely determined by σ. When two rules applies to the same expression e in ML, they apply to nonoverlapping subexpressions. Hence there are no *critical pairs*, i.e. pairs of rules that can apply to the same expression. By *The Critical Pair Lemma* (Terese, 2003, pp. 54–55, Baader & Nipkow, 1998, pp. 139–140), the system is confluent. □

8.2. Complex DC systems. Complex direct constant μ systems, CDC systems, differ from SDC systems in that the set S contains more than one semantic function. Cf. §2.3.

(CDC) *A CDC rule system R*:

i) R_Φ has a finite number of rules;
ii) For any rule r of the μ system, the set of rewrite variables on the rhs of r is a subset of the set of rewrite variables occurring on the lhs of r;
iii) Rewrite variables take all and only OL grammatical terms as instances;
iv) Every atomic rule $r \in R_\Phi$ has the form

$$\mu_i(t) \;\rightarrow\; e,$$

where $\mu_i \in S$, $t \in A_o$ and e is a simple or complex expression in $\mathcal{C}_m$, and for every pair $(\mu_i \in S, t \in A_o)$ there is an atomic rule;
v) Every complex rule $r \in R_\Phi$ has the form

$$\mu_i(\sigma(v_1, \ldots, v_n)) \;\rightarrow\; K(\mu_{k_1}(v_1), \ldots, \mu_{k_n}(v_n)) \;\Leftarrow\; \mathrm{Gr}(\sigma(v_1, \ldots, v_n)),$$

where K is a simple or complex operator of Σ'_m, $\mu_{k_1}, \ldots, \mu_{k_n} \in S$, and $\mathrm{Gr}(\sigma(v_1, \ldots, v_n))$ is the condition that the substitution instance $\sigma(t_1, \ldots, t_n)$ of the application is grammatical; and for every pair (μ_i, σ), $\mu_i \in S$, $\sigma \in \Sigma'_o$, there is a unique complex rule.

The differences between a complex system and the corresponding simple system are in clauses (iv) and (v). According to clause (v) the semantic function symbol μ_{k_j} that will apply to a subterm t_j in the contractum may be different from the semantic function symbol μ_i of the redex, and different from μ_{k_m}, for $j \neq m$. This is what characterizes general compositionality. The proof of the facts below are completely analogous to the proofs of Facts 2 and 3.

PROPOSITION 4. *CDC systems terminate.*

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

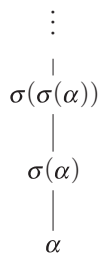

Figure 1. *OL* for **Lisa**.

PROPOSITION 5. *CDC systems are confluent*.

8.3. Indirect systems. Indirect systems are characterized by the fact that complex rules may introduce operators on the rhs that are not terminal symbols, operators in either the set F or G. Such rules are of two kinds, the F kind and the G kind. An F-kind rule represents a noncompositional recursive semantics, and is of the form:

$$\mu(\sigma(v_1, \ldots, v_n)) \rightarrow f(\mu(v_1), \ldots, \mu(v_n), v_1, \ldots, v_n) \quad \Leftarrow \quad \mathrm{Gr}(\sigma(v_1, \ldots, v_n))$$

with $f \in F$. A G-kind rule does not violate compositionality, and has the form:

$$\mu(\sigma(v_1, \ldots, v_n)) \rightarrow g(\mu(v_1), \ldots, \mu(v_n)) \quad \Leftarrow \quad \mathrm{Gr}(\sigma(v_1, \ldots, v_n))$$

with $g \in G$. Accordingly, in both cases there will also have to be a rule for eliminating rewrite terms of the form $f(\ldots)$, or $g(\ldots)$. I shall provide examples. A compositional, G-kind example first.

*8.3.1. The **Lisa** example.* The following example, **Lisa**, is closely related to Davidson's **Annette** example.[28] As in Davidson's case, the language will only contain noun phrases.

$$OL = (\mathcal{T}_o, A_o, \Sigma'_o), \text{ where} \tag{4}$$

i) $A_L = \{\alpha\}$
ii) $\Sigma_o = \{\sigma\}$.

OL is totally ordered by the immediate subterm relation, as shown in Figure 1.

The canonical part of *ML* will have one atomic expressions and two operators. There will also be a member in M:

$$ML = (T_m, A_m, \Sigma'_m, S, F), \text{ where} \tag{5}$$

i) $A_m = \{l\}$
ii) $\Sigma'_m = \{m, h\}$
iii) $S = \{\mu\}$
iv) $F = \{f\}$.

The canonical part of *ML*, $\mathcal{C}_m$, is partially ordered by the immediate subexpression relation, as illustrated in Figure 2.

[28] I presented the **Lisa** example, in a slightly different form, without a fully worked out analysis, at the conference *New Aspects of Compositionality*, in Paris in June 2004. It subsequently appeared in Cohnitz (2005).

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

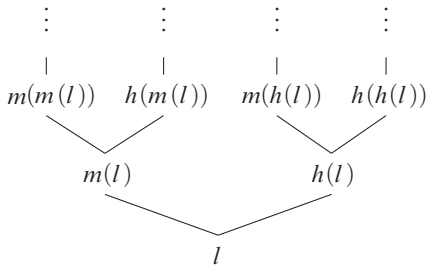

Figure 2. *ML* for **Lisa**.

We will have $v, v_1, v_2, \ldots$ as OL rewrite variables, and $x, x_1, x_2, \ldots$as ML rewrite variables. The set of rules R_{Lisa} is as follows:

$$\begin{array}{ll} \text{i)} & \mu(\alpha) \rightarrow l \\ \text{ii)} & \mu(\sigma(v)) \rightarrow f(\mu(v)) \\ \text{iii)} & f(l) \rightarrow m(l) \\ \text{iv)} & f(m(x)) \rightarrow h(x) \\ \text{v)} & f(h(x)) \rightarrow m(f(x)). \end{array} \tag{6}$$

A sample derivation in R_{Lisa}, reducing $\mu(\sigma(\sigma(\sigma(\sigma(\alpha)))))$ to normal form:

$$\begin{array}{lcll} \mu(\sigma(\sigma(\sigma(\sigma(\alpha)))))) & \rightarrow & f(\mu(\sigma(\sigma(\sigma(\sigma(\alpha)))))) & \text{rule ii)} \\ & \rightarrow & f(f(\mu(\sigma(\sigma(\sigma(\alpha))))))) & \text{rule ii)} \\ & \rightarrow & f(f(f(\mu(\sigma(\sigma(\alpha))))))) & \text{rule ii)} \\ & \rightarrow & f(f(f(f(\mu(\sigma(\alpha))))))) & \text{rule ii)} \\ & \rightarrow & f(f(f(f(l)))) & \text{rule i)} \\ & \rightarrow & f(f(f(m(l)))) & \text{rule iii)} \\ & \rightarrow & f(f(h(l))) & \text{rule iv)} \\ & \rightarrow & f(m(f(l))) & \text{rule v)} \\ & \rightarrow & h(f(l)) & \text{rule iv)} \\ & \rightarrow & h(m(l)) & \text{rule iii)} \end{array} \tag{7}$$

As is illustrated in the example, derivations are longer in relation both to the size of the input term and, even more, in relation to the size of the normal form.

The **Lisa** system is compositional. Yet the output complexity C_L^o of the system is high:

PROPOSITION 6. $C_L^o(k) \geq 2^{k+1} - 2k + 1$.

The proof is given in Appendix B.

The system thus has exponential output complexity, and so is intractable. The result agrees with the outcome of abstract reasoning in §5 about the difference in growth rate between the domain of OL terms and the domain of ML expressions. The input complexity is linear, $C_L^i(k) \simeq 2k$, because of the corresponding size of the input term, again in accordance with §5.[29]

[29] It can be noted that the rule system represents a semantic function only for certain domains. For, if $m(l) = h(l)$, then we have

$$m(l) = h(l) = f(m(l)) = f(h(l)) = m(f(l)) = m(m(l)),$$

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

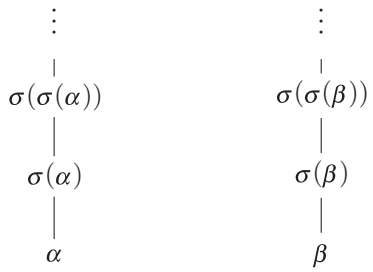

Figure 3. *OL* for **Polly**.

*8.3.2. The **Polly** example.* For a noncompositional example, consider **Polly**.

$$OL = (\mathcal{T}_o, A_o, \Sigma'_o), \text{ where} \tag{8}$$

i) $A_o = \{\alpha, \beta\}$
ii) $\Sigma'_o = \{\sigma\}$.

OL here is a forest of two linear trees, ordered by the immediate subterm relation, as shown in Figure 3.

The *ML* is the same as in **Lisa**, but the rules differ somewhat:

$$\begin{aligned}
&\text{i)}\ \ \mu(\alpha) \rightarrow l \\
&\text{ii)}\ \ \mu(\beta) \rightarrow l \\
&\text{iii)}\ \ \mu(\sigma(v)) \rightarrow f(\mu(v), v) \\
&\text{iv)}\ \ f(x, \sigma(v)) \rightarrow f(x, v) \\
&\text{v)}\ \ f(x, \alpha) \rightarrow m(x) \\
&\text{vi)}\ \ f(x, \beta) \rightarrow h(x).
\end{aligned} \tag{9}$$

That this system is noncompositional is quickly demonstrated:

$$\mu(\alpha) \rightarrow l \quad \text{and} \quad \mu(\beta) \rightarrow l, \quad \text{but} \tag{10}$$

a. $\mu(\sigma(\alpha)) \rightarrow f(\mu(\alpha), \alpha) \rightarrow f(l, \alpha) \rightarrow m(l)$
b. $\mu(\sigma(\beta)) \rightarrow f(\mu(\beta), \beta) \rightarrow f(l, \beta) \rightarrow h(l)$.

For the output complexity C_P^o of **Polly**, where $|x|$ is the size of the term x, notice that the elimination of f from $f(\mu(t), t)$ requires $|t|$ steps, where the last is an application of rule v) or vi), and the preceding steps are applications of rule iv). Hence, the number of steps required to normalize a term $\mu(\sigma(t))$ is $1 + |t| + k$, where k is the number of steps needed normalize the term $\mu(t)$. Since in the **Polly** example, the output size equals the input size, this gives us:

$$C_P^o(k) = \sum_{i=1}^{k} i + 1 = \frac{(k+1)\cdot(k+2)}{2}.$$

The complexity of **Polly** is therefore polynomial. **Polly** is cumbersome but tractable. The reason it is not intractable depends on the fact that it only has a unary

using the assumed identity and rules iii)–v), and if in fact $m(l) \neq m(m(l))$, then there just is no function represented by the function symbol 'f'.

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

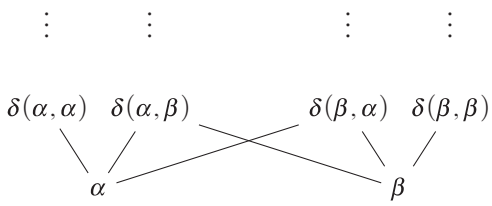

Figure 4. *OL* for **Godzilla**.

operator. Adding a binary operator, the complexity turns intractable, as we shall see next.

*8.3.3. The **Godzilla** example.* The **Godzilla** example is like the **Polly** example, except that we replace the unary syntactic operator σ with the binary operator δ and add a binary operator O to the *ML* (Figure 4).

$$OL = (\mathcal{T}_o, A_o, \Sigma'_o), \text{ where} \tag{11}$$

i) $A_L = \{\alpha, \beta\}$
ii) $\Sigma_o = \{\delta\}$.

The *ML* has a binary function O and two unary functions h and m. The rule system is as follows:

$$\begin{array}{ll}
\text{i)} & \mu(\alpha) \rightarrow l \\
\text{ii)} & \mu(\beta) \rightarrow l \\
\text{iii)} & \mu(\delta(u,v)) \rightarrow f(\mu(u), \mu(v), u, v) \\
\text{iv)} & f(x,y,u,\delta(v_1,v_2)) \rightarrow O(f(x,y,u,v_1), f(x,y,u,v_2)) \\
\text{v)} & f(x,y,u,\alpha) \rightarrow m(f'(x,y,u)) \\
\text{vi)} & f(x,y,u,\beta) \rightarrow h(f'(x,y,u)) \\
\text{vii)} & f'(x,y,\delta(v_1,v_2)) \rightarrow O(f'(x,y,v_1), f'(x,y,v_2)) \\
\text{viii)} & f'(x,y,\alpha) \rightarrow m(x,y) \\
\text{ix)} & f'(x,y,\beta) \rightarrow h(x,y).
\end{array} \tag{12}$$

A sample derivation:

$$\begin{array}{lll}
\mu(\delta(\delta(\alpha,\beta),\delta(\beta,\alpha))) & \rightarrow & f(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \delta(\alpha,\beta), \delta(\beta,\alpha)) \\
 & & \text{rule iii)} \\
 & \rightarrow & O(f(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \delta(\alpha,\beta), \beta), \\
 & & f(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \delta(\alpha,\beta), \alpha)) \quad \text{rule iv)} \\
 & \rightarrow & O(m(f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \delta(\alpha,\beta))), \\
 & & f(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \delta(\alpha,\beta), \alpha)) \quad \text{rule v)} \\
 & \rightarrow & O(m(f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \delta(\alpha,\beta))), \\
 & & h(f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \delta(\alpha,\beta)))) \text{ rule vi)} \\
 & \rightarrow & O(m(O(f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \alpha), \\
 & & f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \beta))), \\
 & & h(f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \delta(\alpha,\beta)))) \text{ rule vii)} \\
 & \rightarrow & O(m(O(f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \alpha), \\
 & & f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \beta))), \\
 & & m(O(f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \alpha), \\
 & & f'(\mu(\delta(\alpha,\beta)), \mu(\delta(\beta,\alpha)), \beta)))) \text{ rule vii)}
\end{array} \tag{13}$$

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

$$
\begin{aligned}
\rightarrow\ & O(m(O(m(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha))),\\
& f'(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha)),\beta))),\\
& m(O(f'(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha)),\alpha),\\
& f'(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha)),\beta)))) \quad \text{rule viii)}\\
\rightarrow\ & O(m(O(m(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha))),\\
& h(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha))))),\\
& m(O(f'(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha)),\alpha),\\
& f'(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha)),\beta)))) \quad \text{rule ix)}\\
\rightarrow\ & O(m(O(m(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha))),\\
& h(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha))))),\\
& m(O(m(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta))),\\
& f'(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha)),\beta)))) \quad \text{rule viii)}\\
\rightarrow\ & O(m(O(m(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha))),\\
& h(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha))))),\\
& m(O(m(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta))),\\
& h(\mu(\delta(\alpha,\beta)),\mu(\delta(\beta,\alpha)))))). \quad \text{rule ix)}
\end{aligned}
$$

The pattern is clear. Reducing a term with the μ argument $\delta(\delta(\alpha,\beta),\delta(\beta,\alpha))$ to a term with μ arguments $\delta(\alpha,\beta)$ and $\delta(\beta,\alpha)$ requires ten steps, as opposed to one in a compositional system, and multiplies the μ terms by four. Hence, the number of steps needed for the two remaining μ terms must be multiplied by four. A lower bound of the input complexity C_G^i of **Godzilla** can be calculated accordingly:

PROPOSITION 7. *$C_G^i(k)$ is factorial.*

The proof is in Appendix C.

The *output* time complexity of **Godzilla** is only polynomial, because of the fact that the canonical expression grows at almost the same rate as the length of the derivation, which illustrates the need to take both input and output complexity into account. Nevertheless, we can provide a variant that also keeps the output size low, and thereby has high output time complexity. These are the rules of the **Godzilla**$^+$ example.

$$
\begin{aligned}
&\text{i)}\quad \mu(\alpha) \rightarrow l\\
&\text{ii)}\quad \mu(\beta) \rightarrow l\\
&\text{iii)}\quad \mu(\delta(u,v)) \rightarrow U(f(\mu(u),\mu(v),u,v))\\
&\text{iv)}\quad f(x,y,u,\delta(v_1,v_2)) \rightarrow O(h(x,y,u,v_1),\ f(x,y,u,v_2))\\
&\text{v)}\quad f(x,y,u,\alpha) \rightarrow f'(x,y,u)\\
&\text{vi)}\quad f(x,y,u,\beta) \rightarrow f'(x,y,u)\\
&\text{vii)}\quad f(x,y,\delta(v_1,v_2)) \rightarrow O(f'(x,y,v_1),\ f'(x,y,v_2))\\
&\text{viii)}\quad f'(x,y,\alpha) \rightarrow m(x,y)\\
&\text{ix)}\quad f'(x,y,\beta) \rightarrow h(x,y)\\
&\text{x)}\quad O(m(v),u) \rightarrow v\\
&\text{xi)}\quad O(h(v),u) \rightarrow u.
\end{aligned}
\tag{14}
$$

The effects of the last two rules is to delete occurrences of 'O' and one of its arguments, which step by step reduces the canonical expression to about the same size as the input term (partly because the rules v) and vi) are different from the corresponding rules in **Godzilla**). In this system, 'O' is not a terminal symbol. Because of the extra

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

deletion steps, the derivation length becomes even greater than in the **Godzilla** example itself.

8.3.4. The general case. The general format of a *complex indirect constant* μ system is as follows:

(CIC) *A CIC rule system* R_Φ:

i) R_Φ has a finite number of rules;
ii) For any rule r of the μ system, the set of rewrite variables on the rhs of r is a subset of the set of rewrite variables occurring on the lhs of r;
iii) Rewrite variables $v_1, v_2, \ldots$ take all and only OL grammatical terms as instances; rewrite variables $y_1, y_2, \ldots$ take all and only expressions in $\mathcal{C}_m$ as instances;
iv) Every atomic rule $r \in R_\Phi$ has the form
- $\mu_i(t) \rightarrow e$,

where $\mu_i \in S$, $t \in A_o$ and e is a simple or complex expression in $\mathcal{C}_m$, and for every pair $(\mu_i \in S, t \in A_o)$ there is an atomic rule;
v) Every *direct* complex rule $r \in R_\Phi$ has the form
- $\mu_i(\alpha(v_1, \ldots, v_n)) \rightarrow K(\mu_{k_1}(v_1), \ldots, \mu_{k_n}(v_n)) \Leftarrow \mathrm{Gr}(\alpha(v_1, \ldots, v_n))$

K is a simple or complex operator of Σ'_m, $\mu_{k_1}, \ldots, \mu_{k_n} \in S$, and $\mathrm{Gr}(\alpha(v_1, \ldots, v_n))$ is as above;
vi) Every *indirect complex input rule* in R_Φ has the form
- $\mu_i(\alpha(v_1, \ldots, v_j)) \rightarrow f(\mu_{k_1}(v_1), \ldots, \mu_{k_n}(v_n), v_1, \ldots, v_n) \Leftarrow \mathrm{Gr}(\alpha(v_1, \ldots, v_n))$

where the rhs arguments $v_1, \ldots, v_n$ are optional and $f \in F$.
vii) For every pair (μ_i, α), $\mu_i \in S$, $\alpha \in \Sigma'_o$, there is a unique complex rule, either a direct rule or an indirect input rule.
viii) Every *indirect ground rule* in R_Φ is of the F-form or the G-form:
- $f(y_1, \ldots, y_n, t_1, \ldots, t_n) \rightarrow K'(f'(y_1, \ldots, y_n))$
- $g(e_1, \ldots, e_n) \rightarrow K(e_1, \ldots, e_n)$,

where $f, f' \in F, g \in G, K, K'$ are operators over Σ'_m, $e_1, \ldots, e_n \in A_m$ and $t_1, \ldots, t_j \in A_o$, and at most one of the primed operators may be null.
ix) Every *indirect recursive rule* in R_Φ has an F- form or a G-form:
- $f(y_1, \ldots, y_n, \sigma(v_1, \ldots, v_n)) \rightarrow K(f(y_1, \ldots, y_n, v_1), \ldots, f(y_1, \ldots, y_n, v_n))$
- $g(K(y_1, \ldots, y_n)) \rightarrow K'(g'(y_1, \ldots, y_n))$,

where $f, \in F, g, g' \in G, K, K'$ are operators over Σ'_m, $\sigma \in \Sigma'_0$, and at most one of the primed operators may be null.

PROPOSITION 8. *CIC systems terminate.*

The proof in is given in Appendix D.

PROPOSITION 9. *CDC systems are confluent.*

Proof. This follows again from the critical pair lemma. □

§9. Compositionality and complexity. We have seen in the **Godzilla** example that the μ-system corresponding to a noncompositional recursive semantics is intractable, since it has a factorial input complexity. In the **Godzilla**$^+$ example, we get a corresponding

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

output complexity as well. The general case of recursive semantics is therefore intractable with a complexity function that is at least factorial.

It is nevertheless not the case that a noncompositional recursive semantics forces exponential complexity. The **Polly** example is noncompositional but has polynomial complexity. This is because the **Polly** OL only has a *unary* syntactic operator. Adding a binary operator leads back to intractable complexity. But it does so only if we allow free embedding of binary (or higher arity) operators under each other. We can retain tractable complexity with binary operators if there are syntactic restrictions that forbids such embeddings completely, or limit them to a fixed finite number of levels, or restrict them to one argument place (other argument places require atomic terms, say[30]), or apply some combination of such syntactic limitations.

Thus we can say that blocking semantic intractable complexity in a *syntax independent* way requires the semantics to be (general) compositional. That is, compositionality is, in this sense, a *necessary* condition for the semantics to be tractable. In a stronger respect, it is a necessary condition for semantics to be minimal, in the sense of having *linear* complexity, for even with only unary operators, as in the **Polly** example, complexity is nonlinear.

Compositionality is not, however, a *sufficient* condition for minimal complexity, or even for tractability. Because in the **Lisa** example, the extra recursion is of the *G*-type, over expressions in the ML, not over terms in the OL, the **Lisa** example is compositional, yet it still has exponential output complexity.[31]

Nevertheless, since (general) compositionality is a consequence of minimal complexity, and in a weaker sense a consequence of tractable complexity, it is a *necessary condition* for an indispensable property of a semantics. This provides a theoretical desideratum that a language to be used by humans be compositional, and an abstract theoretical reason to believe that human languages actually *are* (general) compositional.

To put this conclusion in perspective, it should be noted that we have here only studied the purely semantic part of the comprehension process, i.e. the step from representation of constituent structure to representation of meaning, leaving out two others: the step from surface structure to constituent structure, i.e. parsing, and the integration of the meaning representation into a mental model of a state of affairs. Both these parts of the overall process have cognitive difficulties that are independent of the semantic part, but in fact the relative simplicity of the semantic part provides some help with the others.

The difficulties of parsing emerge in syntactic ambiguity, especially in the embedded case of garden path sentences, where the more natural incremental parsing result fails to fit a particular continuation of the sentence. Compare

(15)
a. While Anna was dressing the child played in the crib.
a. While Anna was dressing the neighbor played the violin.

In (15a), there is a tendency to first take 'the child' as the direct object of 'was dressing', while the continuation of the sentence requires it to be the subject of the

[30] This was suggested by a reviewer.

[31] But note that if compositionality were only a *sufficient* condition for low complexity, we would not automatically get a justification of it, for the question would remain why compositionality would be preferable to the satisfaction of *alternative* sufficient conditions.

matrix clause. This is the garden path effect. The effect is much weaker in (15b), because of the implausible state of affairs that Anna was dressing the neighbor. This makes it less natural to read 'the neighbor' as direct object. The strengthening of the garden path effect in (15a) and the reduction in (15b) depends on the relative ease and speed of the semantic interpretation of the subordinate clause, due to the compositionality of its semantics.[32]

The difficulties of representation of states of affairs arise in cases where multiple simultaneous dependencies. The notorious problems of center-embedding constructions are of this kind. Consider

Dogs [that] dogs [that] dogs attack attack attack. (16)

Without the bracketed complementizers parsing is harder, but the basic problem is not one of parsing, since it does not get much easier when constituent structure is made explicit. The difficulty is rather that of keeping the three nested levels of restriction dependence in the meaning itself in mind in order to create a mental model of the state of affairs. It is not a problem of semantic interpretation. Rather, the fact that there is a straightforward pattern of semantic interpretation allows us to exploit the linguistic articulation as a tool for managing to entertain the proposition in the first place.

Acknowledgments. In memory of Theo Janssen. This paper has been long in the making. Forerunners have been presented at the *Compositionality* conference at Institut Jean-Nicod, Paris 2004; at the *Compositionality Sessions* at Rutgers University 2007; at the inauguration conference of CeLL, Institute of Philosophy, London 2009; at the University of Barcelona 2010; at the *ESSLLI Summer School*, Copenhagen 2010; at Oslo University 2010; at the *Symposium on Language Acquisition and Language Evolution*, at The Royal Swedish Academy of Sciences (KVA), Stockholm 2011; at *The Nordic Logic Summer School*, Stockholm 2017; at the *Gothenburg-Stockholm Workshop on Proof Theory, Model Theory, and Probability*, Gothenburg 2018. I am grateful to more members than I can mention of the audiences at those occasions.

I have benefited especially over the years from comments by and discussing with Ken Chan, Daniel Cohnitz, Östen Dahl, Tim Fernando, Kathrin Glüer, Val Goranko, Theo Janssen, Martin Jönsson, Angelica Kratzer, Shalom Lappin, Aarne Ranta, Tor Sandqvist, Jakub Szymanik, Alasdair Urquhart, Johan van Benthem, Peter van Emde Boas, and Dag Westerståhl. The final text is improved thanks comments from an anonymous referee. The research has been supported by grants from the Tercentenary Foundation of the Swedish National Bank (Riksbankens jubileumsfond), for the project *Interpretational Complexity* and from The Swedish

[32] In Baggio, van Lambalgen, & Hagoort (2012), the authors argue that the impact of on interpretation of plausibility assessments ("world knowledge") is in conflict with compositionality, since compositionality requires that only syntax and lexical meaning be the factors of interpretation. Garden path sentences is one type of example where, according to them, problems of this kind show up. (15a) is their example. They further argue that the only way to save compositionality is to bake world knowledge into lexical meaning, thereby forcing an increase in stored information, which is again bad.

I find their reasoning completely misguided. That other factors make one or other interpretation preferable, among the available interpretations in cases of ambiguity, or that other factors *complement* semantic interpretation in various ways, as in pragmatic enrichment, is simply irrelevant to the truth of the principle. It is not a principle about processing.

Research Council (Vetenskapsrådet) for the project *Compositionality*. The project has also received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 675415.

§A. Appendix: Proof of Proposition 1. *Representation theorem*

Given a semantic algebra SA_{LM}, there is a semantic algebra ordering $\prec$ of $GT_L \cup M$ and a Gödel numbering $\ulcorner\cdot\urcorner$ from $GT_L \cup M$ to N, to the effect that for any recursive$^s_\prec$ function $f : (GT_L \cup M)^n \longrightarrow GT_L \cup M$ there is a function $\overline{f} : N^n \longrightarrow N$ such that

a) for any $x_1, \ldots, x_n \in GT_L \cup M$, $\ulcorner f(x_1, \ldots, x_n)\urcorner = \overline{f}(\ulcorner x_1\urcorner, \ldots, \ulcorner x_n\urcorner)$,
b) $\overline{f}$ is recursive.

Proof. Let $\ulcorner\cdot\urcorner$ be a Gödel numbering satisfying the condition that

(i) for each $\delta \in \Sigma_L \cup R_M$, there is a strictly monotone increasing arithmetic function $\overline{\delta}$ such that $\ulcorner\delta(x_1, \ldots, x_n)\urcorner = \overline{\delta}(\ulcorner x_1\urcorner, \ldots, \ulcorner x_n\urcorner)$.

Condition (i) can be satisfied. For instance, an adaptation of the Gödel numbering method given in Boolos *et al.* (2002), chap. 15, does meet it. Each simple term-building symbol e is assigned a value $\ulcorner e\urcorner$. Where $10^k \leq i < 10^{k+1}$, let $\widetilde{x_i} = k + 1$. Then, as part of the definition of $\ulcorner\cdot\urcorner$, we let

$$\ulcorner\delta(x_1, \ldots, x_n)\urcorner = \ulcorner\delta\urcorner \cdot 10^{\widetilde{x_1}+\cdots+\widetilde{x_n}} + \ulcorner x_1\urcorner \cdot 10^{\widetilde{x_2}+\cdots+\widetilde{x_n}} + \cdots + \ulcorner x_{n-1}\urcorner \cdot 10^{\widetilde{x_n}} + \ulcorner x_n\urcorner.$$

(that is, in decimal notation we get the codes for complexes by writing the codes of the constituents in sequence from left to right). Clearly, condition (i) is met. We now define $\prec$ such that $t \prec t'$ iff $\ulcorner t\urcorner < \ulcorner t'\urcorner$.

It is easily seen that conditions of Definition 8 are met. For complex terms, the term with the greater $|\cdot|$ size has the greater $\ulcorner\cdot\urcorner$ number. For terms of equal size, the value of the main operator is most significant, and after that the arguments from left to right, both for $\prec$ between terms and for $<$ between Gödel numbers.

Now we can check that any for any recursive$^s_\prec$ function h, there is a function $\overline{h}$ satisfying a) and b):

i) h is a constant basic function: since for each natural number k there is a recursive constant function with k as value, let $\overline{h}$ be a constant function such that for any argument $\ulcorner x\urcorner$, $\overline{h}(\ulcorner x\urcorner) = \ulcorner h(x)\urcorner$.
ii) h is a function $\delta \in \Sigma_L \cup R_M$. By (i) we let $\overline{h}$ be $\overline{\delta}$. Hence a) is satisfied. Since $\overline{h}$ is an arithmetic function, it is recursive, and so b) is met.
iii) h is a projection function. Then let $\overline{h}$ be a corresponding projection function. a) and b) are immediately satisfied.
iv) h is defined from $f, g_1, \ldots, g_n$ by function composition, where $f, g_1, \ldots, g_n$ are recursive$^s_\prec$ and we can assume as induction hypothesis that there are $\overline{f}, \overline{g_1}, \ldots, \overline{g_n}$ that satisfy conditions a) and b). Let $\overline{h}$ be defined such that

$$\overline{h}(\ulcorner x_1\urcorner, \ldots, \ulcorner x_n\urcorner) = \overline{f}(\overline{g_1}(\ulcorner x_1\urcorner, \ldots, \ulcorner x_n\urcorner), \ldots, \overline{g_n}(\ulcorner x_1\urcorner, \ldots, \ulcorner x_n\urcorner)).$$

It immediately follows that $\overline{h}$ is recursive. It is also easy to verify, using the induction hypothesis, that condition a) is met.

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

v) h is defined by primitive recursion from functions f_y and g_δ:

$$h(\overrightarrow{x}, y) = \begin{cases} f_y(\overrightarrow{x}), & \text{if } y \in AT_L \cup B_M \\ g_\delta(h(\overrightarrow{x}, y_1), \dots, h(\overrightarrow{x}, y_k), \overrightarrow{x}, y_1, \dots, y_k), & \text{if } y = \delta(y_1, \dots, y_k), \end{cases}$$

where it is assumed that for each relevant g_δ and f_y there are functions $\overline{g_\delta}$ and $\overline{f_y}$, respectively, that satisfy conditions a) and b).

Then we define a function $\overline{h}$ by cases:

$$\overline{h}(k_1, \dots, k_n, k) = \begin{cases} \text{undefined}, & \text{if } h \text{ undefined, or } k_1, \dots, k_n \text{ or } k \text{ is not a code} \\ \overline{f_y}(\overrightarrow{\ulcorner x \urcorner}), & \text{if } k = \ulcorner y \urcorner, k_i = \ulcorner x_i \urcorner,\ y \in AT_L \cup B_M \\ \overline{g_\delta}(\overline{h}(\overrightarrow{\ulcorner x \urcorner}, \ulcorner y_1 \urcorner), \dots, \overline{h}(\overrightarrow{\ulcorner x \urcorner}, \ulcorner y_k \urcorner), \overrightarrow{\ulcorner x \urcorner}, \ulcorner y_1 \urcorner, \dots, \ulcorner y_k \urcorner), & \\ \qquad \text{if } k = \ulcorner y \urcorner, k_i = \ulcorner x_i \urcorner,\ y = \delta(y_1, \dots, y_k). \end{cases}$$

Here $1 \leq i \leq n$. Since definition by cases is recursively definable, $\overline{h}$ is (partial) recursive.

It is almost immediate from the induction hypothesis that condition a) is met in the second case of the definition of $\overline{h}$. For the third case, we use induction over the complexity of the y argument, and use the induction hypothesis to show that condition a) is met in the induction step. That completes both induction proofs, and condition a) is met.

vi) h is $\mathrm{Mn}_\prec(f)$, where we can assume that there is a function $\overline{f}$ that meets conditions a) with respect to f and is recursive. We first define a function f' from $\overline{f}$ such that if

$$\overline{f}(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner, \ulcorner x \urcorner) = \ulcorner @ \urcorner,$$

$\overline{f}$ is defined for $\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner, \ulcorner x' \urcorner$, for all $\ulcorner x' \urcorner < \ulcorner x \urcorner$, and for no $\ulcorner x' \urcorner < \ulcorner x \urcorner$ does it hold that $\overline{f}(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner, \ulcorner x' \urcorner) = \ulcorner @ \urcorner$, then

$$f'(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner, \ulcorner x \urcorner)) = 0.$$

We let $f'(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner, \ulcorner x \urcorner)) = 1$ if $\overline{f}(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner, \ulcorner x \urcorner))$ is undefined, and otherwise $f'(k_1, \dots, k_n, i)) = \overline{f}(k_1, \dots, k_n, i)$. We then define a partial function $\overline{h} : \mathbb{N}^n \longrightarrow \mathbb{N}$, such that,

a) if $h(x_1, \dots, x_n)$ is undefined, then $\overline{h}(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner)$ is undefined;
b) if $h(x_1, \dots, x_n)$ is defined, then

$$\overline{h}(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner) = \mathrm{Mn}(f')(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner).$$

Now, f' is definable from $\overline{f}$ by cases in terms of $\ulcorner @ \urcorner$. $\overline{f}$ is recursive by assumption, so f' is recursive.

Then we can verify that $\overline{h}$ satisfies condition a) with respect to h:

$$\begin{aligned} \overline{h}(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner) = \ulcorner x \urcorner \quad \text{iff} \quad & f'(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner, \ulcorner x \urcorner) = 0 \text{ and } C \\ \text{iff} \quad & \overline{f}(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner, \ulcorner x \urcorner) = \ulcorner @ \urcorner \text{ and } C' \\ \text{iff} \quad & \ulcorner f(x_1, \dots, x_n, x) \urcorner = \ulcorner @ \urcorner \text{ and } C'' \\ \text{iff} \quad & \ulcorner h(x_1, \dots, x_n) \urcorner = \ulcorner x \urcorner, \end{aligned}$$

where

i) C iff $f'(\ulcorner x_1 \urcorner, \dots, \ulcorner x_n \urcorner, \ulcorner x' \urcorner)$ is defined but $\neq 0$ for all $\ulcorner x' \urcorner < \ulcorner x \urcorner$,

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

ii) C' iff @ is $\prec$-minimal, $\overline{f}(\ulcorner x_1 \urcorner, \ldots, \ulcorner x_n \urcorner, \ulcorner x' \urcorner)$ is defined and $\neq \ulcorner @ \urcorner$ for all $\ulcorner x' \urcorner < \ulcorner x \urcorner$,

iii) C'' iff @ is $\prec$-minimal, $\ulcorner f(x_1, \ldots, x_n, x') \urcorner$ is defined and $\neq \ulcorner @ \urcorner$ for all $x' \prec x$.

It can be verified that C iff C' iff C'', because of the construction of f', because of the induction hypothesis concerning f and $\overline{f}$, and because of the definitions of $\ulcorner \cdot \urcorner$, $\prec$, and @. □

§B. Appendix: Proof of Proposition 6. $C_L^o(k) \geq 2^{k+1} - 2k + 1$.

Proof. An inspection of the rules of (4) shows that only rules i) and iii) produce a new canonical symbol occurrence, and that only a term of the form $h(f(\ldots f(l)))$, by repeated applications of rule v) leads to an application or rule iii). We first show that for a term of the form

$$f(f(\ldots f(m(m \ldots m(x)))))) \tag{17}$$

with i occurrences of 'f' and k occurrences of 'm' outside x to be reduced, by means of alternating applications or rules iv) and v) to

$$h(\ldots h(x))$$

with k occurrences of 'h' outside x, i must be at least $2^k - 1$. We show this by induction over k. It does hold for $k = 1$ (i.e. $f(m(x))$). Suppose it holds for $k = n$. For $k = n + 1$, note that a term t of the form (17) with $n + 1$ occurrences of m outside x is also a term t' of the form (17) with n occurrences of m outside x', where $x' = m(x)$. By the induction hypothesis, therefore, $2^n - 1$ occurrences of f are required to reduce t' to the term u

$$u = f(f(\ldots h(h(\ldots m(x) \ldots)) \ldots))$$

with n occurrences of h outside x. Now, by successively applying rule v) to the outermost occurrence of h, in n steps we arrive at term u'

$$u' = f(f(\ldots m(m(\ldots f(m(x))))))$$

with n occurrences of m outside x. This lets us apply the induction hypothesis again, again requiring $2^n - 1$ occurrences of f for reduction to a term s

$$s = f(f(\ldots h(h(\ldots f(m(x) \ldots)) \ldots)))$$

with n occurrences of h outside x. A final application of rule iv) to the subterm $f(m(x))$ transforms it to $h(x)$, leaving us with the term s'

$$s' = f(f(\ldots h(h(\ldots (x)) \ldots)))$$

with $n + 1$ occurrences of h outside x. To achieve this we needed $2 \times (2^n - 1) + 1 = 2^{n+1} - 1$ occurrences of f outside x, which completes the induction step.

Second, we note that a term of the form with $f(f(\ldots f(m(m \ldots m(l)))))$ with $k + 1$ occurrences of m results from k applications of rule v) and one application of rule iii) to a term $f(f(\ldots f(h(h \ldots h(l)))))$ with k occurrences of h. This term must then first be generated, from the corresponding term $f(f(\ldots f(m(m \ldots m(l)))))$ with k occurrences of m, and so on. This means that the total number of occurrences of f required for

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

generating a term $f(f(\ldots f(h(h\ldots h(l)))))$ with k occurrences of h is

$$\sum_{i=1}^{k} 2^i - 1.$$

For each occurrence of f, at least one application of rule ii) is needed to generate it, and at least one application of rule iv) or rule iii) is needed to eliminate it. Hence, the number of derivation steps will be at least

$$\sum_{i=1}^{k} 2^{i+1} - 2 \quad = \quad 2^{k+2} - 2k - 1,$$

where k is the number of occurrences of terminal symbols except l. Translated into the number of terminal symbols simpliciter, including l, we have

$$C_L^o(k) \geq 2^{k+1} - 2k + 1.$$

For $k \geq 3$, the value will be higher, because of applications of rule v), which neither introduces nor eliminates any occurrence of f. But it will not be higher by an order of magnitude. □

§C. Appendix: Proof of Proposition 7. $C_G^i(k)$ is factorial.

Proof. Let $C(x)$ be the number of steps needed to normalize the term x. We can see that rules iv) and vii) have the following effects:

$$C(f(x,y,u,\delta(t_1,t_2))) = 1 + C(f(x,y,u,t_1)) + C(f(x,y,u,t_{t_2}))$$
$$C(f'(x,y,\delta(s_1,s_2))) = 1 + C(f'(x,y,s_1)) + C(f'(x,y,s_2)).$$

Let $D(t)$ be the sum of occurrence of α and β in t. We can see from the two equations above that, for $t = \delta(t_1,t_2)$, $C(\mu(t)) \geq (C(\mu(t_1))) + C(\mu(t_2))) \times D(t)$. This is because the processing of the subterms $\mu(t_1)$ and $\mu(t_2)$ will be multiplied once for each atomic subterm in t. And this pattern will be repeated for each level of the syntactic term, i.e. the depth of embedding of δ.

This motivates the hypothesis

$$C_G^i(2k-1) \geq (k-1)!, \tag{18}$$

where k is the number of occurrences atomic terms, i.e. α and β. We prove this by means of induction over k.

Base step. For $k = 1$, the number of steps needed is 1, an application of either rule i) or rule ii), for the atomic cases. According to the hypothesis, $C_G^i(2 \cdot 1 - 1) = C_G^i(1) \geq (1-1)! = 1$.

For the induction step, suppose that the hypothesis is true for $k = n$. Assume that we have a term $t = \delta(t_1,t_2)$ with $n+1$ occurrences of atomic subterms. An application of rule iii) leads to the ML term

$$f(\mu(t_1),\mu(t_2),t_1,t_2).$$

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

Now, let $N(\delta, t_i)$ be the number of occurrences of δ in t_i, $i = 1, 2$. In case t_2 is atomic, $N(\delta, t_2) = 0$ and in case t_1 is atomic, $N(\delta, t_2) = n - 1$. Observe that in any term in the OL, the number of occurrences of atomic terms is one higher than the number of occurrences of δ.

Assume $N(\delta, t_2) = 0$. The next step is an application of rule v) or rule vi) depending on whether t_2 is α or β. Since now $N(\delta, t_1) = n - 1$, this step is followed by $n - 1$ applications of rule vii). Each application of rule vii) replaces one occurrence of the terms $\mu(t_1)$ and $\mu(t_2)$ by two occurrences. Hence, the result after $n - 1$ applications is n occurrences of these terms. According to the assumption, t_1 has n occurrences of atomic subterms and t_2 1 occurrence. Thus, the induction hypothesis applies in both cases. This gives us a lower bound on the maximal number $C_0(\mu(t))$ of rule applications for $\mu(t)$ on the assumption that $N(\delta, t_2) = 0$:

$$C_0(\mu(t)) \geq 1 + (n-1) + n((n-1)! + (1-1)!) = n! + 2n \geq n!.$$

Notice that for $0 < j < n/2$, it holds that $(n-j)! + (j-1)! \geq (n-(j+1))! + (j+1-1)!$, provided $n \geq 3$. Therefore, we get the maximal value of $C(\mu(t))$ in case $N(\delta, t_2) = 0$, i.e. in case $C(\mu(t) = C_0(\mu(t)))$. This completes the induction.

Hence, $C_G^i(2k-1) \geq (k-1)!$. Note that the size of an OL term is always odd. In the standard format, we then have $C_G^i(k) \geq \frac{k-1}{2}!$. □

§D. Appendix: Proof of Proposition 8. We shall prove that CIC systems terminate. This is more difficult, since applications of the indirect complex rules can actually increase the size of the term: the contractum will be larger than the redex. Because of this, we shall need a more complex reduction order. We shall define a strict *lexicographic order* $>_{S,F,G}$ from three strict orders $>_S$, $>_F$, and $>_G$. The first will be similar to the size definition we gave for terms in the proof of Proposition 2 and be defined as the sum of the sizes of μ subterms. The other two will be defined on the height of embeddings rather than the size of terms.

Proof. We start by defining $>_S$ by means of the S-size $|s|_S$ of a term in $\mathcal{T}_m$. For the S-size of s, only the total sum of the S-sizes of occurrences μ-terms in s matters:

($>S$) i) for any $t \in \mathcal{T}_o$, $\mu_i \in S$, $|\mu_i(t)|_S = |t|_S$
ii) for any $t \in A_o$, $|t|_S = 1$
iii) for any n-place $\alpha \in \Sigma'_o$, any terms $t_1, \ldots, t_n$, $|\alpha(t_1, \ldots, t_n)|_S = 1 + |t_1|_S + \cdots + |t_n|_S$.
iv) Let s be a term in $\mathcal{T}_m$, and let $s_\mu = \{s_1, \ldots, s_n\}$ be the set of *occurrences* of μ-terms in s. Then $|\mathcal{T}_m|_S = |s_1|_S + \cdots + |s_n|_S$ (where the size of an occurrence is the size of the term that occurs).
v) For $s, u \in \mathcal{T}_m$, $s >_S u$ iff $|s|_S > |u|_S$.

Thus, for a *μ-free* term s in $\mathcal{T}_m$, $|s|_S = 0$.

Next we define $>_F$ by means of the F-measure of a term. The F-measure of a term x is a function s_x from natural numbers to natural numbers such that $s_x(k)$ is the number of F subterms $f(y)$ of x with F-size k. The F-size is defined as syntactic height of terms in $\mathcal{T}_o$:

($>F$) i) for $t \in A_o$, $[t] = 1$

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

ii) for $t \in \mathcal{T}_o$, $t = \sigma(t_1, \dots, t_n)$, $[t] = 1 + \max([t_1], \dots, [t_n])$
iii) for $t \in \mathcal{T}_o$, $[\mu(t)] = 0$
iv) for $e \in A_m$, $[e] = 0$
v) for $e_1, \dots, e_n \in \mathcal{T}_m$, $K \in \Sigma_m$, $[K(e_1, \dots, e_n)] = \max([e_1], \dots, [e_n])$
vi) for $e_1, \dots, e_n \in \mathcal{T}_m$, $t_1, \dots, t_m \in \mathcal{T}_o$, $f \in \Sigma_m$, $[f(e_1, \dots, e_n, t_1, \dots, t_n)] = \max([e_1], \dots, [e_n], [t_1], \dots, [t_m])$
vii) $s_x(j) = k$ iff k is the number of subterms (proper or improper) of x of the form $f(y)$ such that $[y] = j$
viii) $x >_F y$ iff there is n such that $s_x(n) > s_y(n)$ and for all $k > n$, $s_x(k) = s_y(k)$.

Next we define $>_G$ by means of the G-measure of a term. The G-measure of a term x is a function s'_x from natural numbers to natural numbers such that $s'_x(k)$ is the number of G subterms $g(y)$ of x with G-size k. The G-size is defined as syntactic height of terms in $\mathcal{T}_m$:

($>G$) i) for $t \in \mathcal{T}_o$, $[t]' = 0$
ii) for $t \in \mathcal{T}_o$, $[\mu(t)]' = 1$
iii) for $e \in A_m$, $[e]' = 1$
iv) for $e_1, \dots, e_n \in \mathcal{T}_m$, $K \in \Sigma_m$, $[K(e_1, \dots, e_n)]' = 1 + \max([e_1]', \dots, [e_n]')$
v) for $e_1, \dots, e_n \in \mathcal{T}_m$, $g \in G$, $[g(e_1, \dots, e_n)]' = \max([e_1]', \dots, [e_n])'$
vi) $s'_x(j) = k$ iff k is the number of subterms (proper or improper) of x of the form $f(y)$ such that $[y]' = j$
vii) $x >_G y$ iff there is n such that $s'_x(n) > s'_y(n)$ and for all $k > n$, $s'_x(k) = s'_y(k)$.

Induction over term complexity shows that $[\cdot]$ and $[\cdot]'$ are well-defined: for each term x in $\mathcal{T}_m$, both $[x]$ and $[x]'$ are natural numbers. Further, since it is also definite what the occurrences in a term x of subterms of the form $f(y)$ are, $s_x(j)$ and $s'_x(j)$ are well-defined, too. And since for any $e \in \mathcal{T}_m$ there is a number n such that for all $k > n$, $s_x(j) = 0$, the relation $>_F$ is also well-defined. Likewise for $>_G$.

Next we define the lexicographic order $>_{SFG}$:

($>S,F,G$) $x >_{S,F,G} y$ iff

i) $x >_s y$, or
ii) $|x|_S = |y|_S$ and $x >_F y$, or
iii) $|x|_S = |y|_S$, $[x]_F = [y]_F$, and $x >_G y$.

Finally, we check the rule types to see that if $x \to y$, then $x >_{S,F,G} y$:

i) If $x \to y$ because of application of an atomic rule $\mu_i(t) \to e$, then $|x|_S = |y|_S + 1$, since y contains one atomic μ term occurrence less than x and otherwise the same μ terms with corresponding occurrences. Hence, $x >_{S,F,G} y$.
ii) If $x \to y$ because of an application of a direct complex rule, then again $|x|_S = |y|_S + 1$, since y results from x by substituting '$K(\mu_{k_1}(t_1), \dots, \mu_{k_n}(t_n))$' for '$\mu_i(\alpha(t_1, \dots, t_n))$', and $|\mu_i(\alpha(t_1, \dots, t_n))| = 1 + |K(\mu_{k_1}(t_1), \dots, \mu_{k_n}(t_n))|$.
iii) If $x \to y$ because of an application of an indirect complex input rule, we have the same result in (ii).
iv) If $x \to y$ by application of an indirect ground rule, then $|x| = |y|$, but in the first case (where $f(y_1, \dots, y_n, t_1, \dots, t_n) \to K'(f'(y_1, \dots, y_n))$) $[x] > [y]$ and in

https://doi.org/10.1017/S1755020320000027 Published online by Cambridge University Press

the second case (where $f(e_1, \ldots, e_n) \rightarrow K(e_1, \ldots, e_n)$), $|x| = |y|$ and $[x] = [y]$ but $[x]' > [y]'$. In the first case, this is because $s_x(n) > s_y(n)$ (since one f term with n OL term arguments is replaced in the rule application) while $s_x(k) = s_y(k)$ for any $k > n$ (no f term with more OL term arguments is affected). In the second case, we have the corresponding result for s'.

v) We get the corresponding results for applications of indirect recursive rules as in (iv). The first case leaves $|\cdot|$ intact and lowers $[\cdot]$, while the second case leaves both $|\cdot|$ and $[\cdot]$ intact but lowers $[\cdot]'$. Hence, if $x \rightarrow y$ with any CIC rule application $x >_{S,F,G} y$.

□

BIBLIOGRAPHY

[1] Abelard, P. (2010). Logica 'ingredientibus'. 3. Commentary on Aristotele's De Interpretatione. In Jacobi, K. & Straub, C., editors. *Corpus Christianorum Contiunatio Medievalis*. Turnhout: Brepols Publishers.

[2] Baader, F. & Nipkow, T. (1998). *Term Rewriting and All That*. Cambridge: Cambridge University Press.

[3] Baggio, G., van Lambalgen, M., & Hagoort, P. (2012). The processing consequences of compositionality. In Hinzen, W., Machery, E., and Werning, M., editors. *The Oxford Handbook of Compositionality*. Oxford: Oxford University Press, pp. 655–672.

[4] Boolos, G. S., Jeffrey, R., & Burgess, J. P. (2002). *Computability and Logic* (fourth edition). Cambridge: Cambridge University Press.

[5] Boolos, G. S. & Jeffrey, R. C. (1980). *Credibility and Logic* (second edition). Cambridge: Cambridge University Press.

[6] Brighton, H. J. (2005). Linguistic evolution, and induction by minimum description length. In Werning, M., Machery, E., & Schurz, G., editors. *The Compositionality of Concepts and Meanings*: *Applications to Linguistics*, *Psychology and Neuroscience*, Vol. 2. Frankfurt/Lancaster: Ontos, pp. 13–39.

[7] Bunt, H. & Muskens, R. (1999). Computational semantics. In Bunt, H., and Muskens, R., editors. *Computing Meaning*, Vol. 1. Amsterdam: Kluwer Academic Publishers, pp. 1–32.

[8] Buridan, J. & van Lecq, R. (1998). *Summulae de Dialectica* 4. *Summulae de Suppositionibus*, Vol 10. Nijmegen, Netherlands: Artistarium, p. 4.

[9] Cohnitz, D. (2005). Is compositionality an a priori principle? In Werning, M., Machery, E., and Schurz, G., editors. *The Compositionality of Concepts and Meanings*: *Foundational Issues*. Frankfurt/Lancaster: Ontos, pp. 23–58.

[10] Copeland, B. J. (2008). The church-turing thesis. In Zalta, E. N., editor. *The Stanford Encyclopedia of Philosophy*. Stanford, CA: CSLI Publications, Stanford University. Available from: http://plato.stanford.edu/archives/fall2008/entries/church-turing/.

[11] Cummins, R. (1989). *Meaning and Mental Representation*. Cambridge, MA: MIT Press.

[12] Dal Lago, U. & Martini, S. (2008). The weak lambda calculus as a reasonable machine. *Theoretical Computer Science*, **398**, 32–50.

[13] ———. (2009). On constructor rewrite systems and the lambda-calculus. In Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., and Thomas, W., editors. *Automata, Languages and Programming, 36th International Colloquium, Part II*. Lecture Notes in Computer Science, Vol. 5556, London: Springer, pp. 163–174.

[14] Davidson, D. (1965). Theories of meaning and learnable languages. In Bar-Hillel, Y., editor. *Logic, Methodology and Philosophy of Science II*. Amsterdam: North-Holland, pp. 1–15.

[15] ———. (1967). Truth and meaning. *Inquiries into Truth and Interpretation*. Oxford: Clarendon Press, pp. 17–36. Originally published in *Synthèse* (1967), **17**, 304–323.

[16] ———. (1984). *Inquiries into Truth and Interpretation*. Oxford: Clarendon Press.

[17] Fodor, J. (1987). *Psychosemantics*. Cambridge, MA: MIT Press.

[18] Garey, M. R. & Johnson, D. S. (1979). *Computers and Intractability. A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Company.

[19] Glüer, K. & Pagin, P. (2006). Proper names and relational modality. *Linguistics & Philosophy*, **29**, 507–535.

[20] Glüer, K. & Pagin, P. (2008). Relational modality. *Journal of Logic, Language and Information*, **17**, 307–322.

[21] Glüer, K. & Pagin, P. (2012). General terms and relational modality. *Nous*, **46**, 159–199.

[22] Hartmanis, J. & Stearns, R. E. (1965). On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, **117**, 285–306.

[23] Heim, I. & Kratzer, A. (1998). *Semantics in Generative Grammar*. Oxford: Blackwell.

[24] Hendriks, H. (2001). Compositionality and model-theoretic interpretation. *Journal of Logic, Language and Information*, **10**, 29–48.

[25] Hodges, W. (2001). Formal features of compositionality. *Journal of Logic, Language and Information*, **10**, 7–28.

[26] Immerman, N. (2008). Computability and complexity. In Zalta, E. N.. editor. *Stanford Encyclopedia of Philosophy*. Stanford, CA: CSLI Publications, Stanford University. Available from: http://plato.stanford.edu/archives/fall2008/entries/computability/.

[27] Janssen, T. M. V. (1997). Compositionality. In van Benthem, J. & Meulen, A. T., editor. *Handbook of Logic and Language.* Amsterdam: Elsevier, pp. 417–473.

[28] Li, M. & Vitányi, P. (1997). *An Introduction to Kolmogorov Complexity and its Applications* (second edition). New York: Springer.

[29] Magidor, O. (2009). Category mistakes are meaningful. *Linguistics and Philosophy*, **6**, 553–581.

[30] Montague, R. (1973). The proper treatment of quantification in ordinary English. In Hintikka, J., Moravcsik, J., and Suppes, P., editors. *Approaches to Natural Language*. Dordrecht, Netherlands: Reidel, pp. 247–270. Reprinted in Montague 1974, page references to the reprint.

[31] ———. (1974). *Formal Philosophy*: *Selected Papers by Richard Montague*. New Haven, CT: Yale University Press.

[32] Pagin, P. (2003). Communication and strong compositionality. *Journal of Philosophical Logic*, **32**, 287–322.

[33] ———. (2012a). Communication and the complexity of semantics. In Hinzen, W., Machery, E., & Werning, M., editors. *The Oxford Handbook of Compositionality*. Oxford: Oxford University Press, pp. 510–529.

[34] ———. (2012b). Truth-theories, competence, and semantic computation. In Preyer, G., editor. *Davidson's Philosophy*: *A Reappraisal*. Oxford: Oxford University Press.

[35] ———. (2013). Compositionality, complexity, and evolution. In Lacerda, F., editor. *Proceedings from a Symposium on Language Acquisition and Language Evolution*. Stockholm, Sweden: PERILUS Stockholm University, Department of Linguistics. Available from: http://www.ling.su.se/perilus/perilus-2011.

[36] ———. (2019a). Belief sentences and compositionality. notional part. *Journal of Semantics*, **36**, 241–284.

[37] ———. (2019b). Compositionality in Davidson's early philosophy. *Journal for the History of Analytical Philosophy* , **7**(2), 76–89.

[38] Pagin, P. & Westerståhl, D. (2010a). Compositionality I: Definitions and variants. *Philosophy Compass*, **5**, 250–264.

[39] ———. (2010b). Compositionality II: Arguments and problems. *Philosophy Compass*, **5**, 265–282.

[40] ———. (2010c). Pure quotation and general compositionality. *Linguistics and Philosophy*, **33**, 381–415.

[41] Potts, C. (2007). The dimensions of quotation. In Barker, C. & Jacobson, P., editors. *Direct Compositionality*. Oxford: Oxford University Press, pp. 405–431.

[42] Shannon, C. E. (1949). The mathematical theory of communication. In Shannon, C. E. & Weaver, W., editors. *The Mathematical Theory of Communication*. Chicago: The University of Illinois Press.

[43] Terese. (2003). *Term rewriting systems*. Cambridge Tracts in Theoretical Computer Science, Vol. 55. Cambridge: Cambridge University Press.

[44] van Emde Boas, P. (1990). Machine models and simulations. In van Leeuwen, J., editor. *Handbook of Theoretical Computer Science*. Amsterdam: Elsevier, pp. 3–66.

[45] van Rooij, I. (2008). The tractable cognition thesis. *Cognitive Science*, **32**, 939–984.

[46] Werning, M. (2005). Right and wrong reasons for compositionality. In Werning, M., Machery, E., & Schurz, G., editors. *The Compositionality of Concepts and Meanings*: *Foundational Issues*. Frankfurt, Germany/Lancaster: Ontos-Verlag, pp. 285–309.

[47] Werning, M., Machery, E., & Schurz, G., editors. (2005). *The Compositionality of Concepts and Meanings*: *Foundational Issues*. Frankfurt, Germany/Lancaster: Ontos-Verlag, pp. 23–58.

[48] Westerståhl, D. (2004). On the compositional extension problem. *Journal of Philosophical Logic*, **35**, 549–582.

DEPARTMENT OF PHILOSOPHY
STOCKHOLM UNIVERSITY
STOCKHOLM, SWEDEN
E-mail: peter.pagin@philosophy.su.se