0. INTRODUCTION

0.1 A LANGUAGE MACHINE

Amidst efforts to Christianize Muslims, to describe '... profane love in scenes of such repulsive realism that they would shock even an admirer of Henry Miller's fiction' (Gardner 1958:7), to develop voting systems (which are still in use), and to square the circle – his construction approached a circle's surface up to 0.04 % – Ramón Llull invented *logic machines*. In 13th-century Catalonia, he designed and exploited the systematic and mechanical production of propositions. For example, he wrote phrasal concepts on the edges of concentric circles. Connecting the concepts in one system of circles (or disks), by moving the circles with respect to each other, yields propositions with and about these concepts, according to the semantic frame of the particular system. Here is Llull's generator for truisms on the soul (from: *Ars demonstrativa*, c. 1283)– one among hundreds of systems on all fields of philosophical theology.

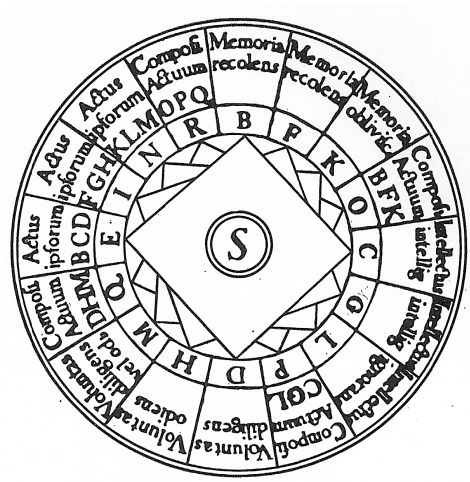

Figure 1 Ramon Llull's Prima figura S

It produces in each state four conjunctions of three 'small clauses', where each conjunction describes a state of the soul, according to Gardner (1958:13) up to a total of 136 combinations. Llull considered these generators as his *Ars Magna*, his major trick. And though Llull contributed little to the material knowledge of his days, Leibniz recognized him as a pioneer in combinatory logic: *let us calculate* instead of endlessly disputing. Llull's disks are grammars, generative grammars of meaningful sentences. Unfortunately, Llull considered his grammars to generate (all and only) true propositions – a quite common overestimation of the power of linguistics. Yet, the idea that propositions are meaningful *because* or *to the extent that* they are generated by a system and that meaningfulness is not an accident, we may grant to the 'obscurantist' Ramon Llull.

Not everyone was and is convinced by Llull's way of working, just as not every scholar of language finds comfort in its mechanization. Llull and Leibniz tried to tell *truth* from *falsehood* – an utterly philosophical and *non-linguistic* enterprise. This book seeks to explore the computability of meaningful language, in order to tell *meaningful* from *meaningless* Dutch – an utterly *linguistic* enterprise. Llull was after the grammar of truth. Unfortunately, there is none. This book is after the grammar of meaningfulness, trying to show that such a grammar can be established and be made to work. It investigates the *modus operandi* of a particular language machine, called Delilah – the biblical connotation is a mere accident. This machine, available at http://www.delilah.eu, executes a language program electronically, inspecting data, computing actions and storing results. It produces meanings that correlate with meaningful sentences, and meaningful sentences correlating with meanings. It can be seen as an act of grammar engineering, an effort to make grammar work and do part of the job of the language processing human brain. It is certainly not an effort to improve on that processing, or to take away the burden of processing. It is meant to model one human language in order to grasp how it is designed and why it works, and in order to be able to test hypotheses on its deepest feature: the relation between form and meaning, the success of arbitrary forms conveying inter-subjective meaningfulness. The machine that we describe and investigate is not an application. It is just a defective, but also instructive and promising model of human language processing. It is a contribution to linguistics-by-computation – it is computational linguistics. Thus, it is neither a 'bachelor machine' – pure design and un-built – like the language machines of Raymond Roussel (Carrouges 1975) – nor just a poem: 'A poem is a (small or) large machine made out of words' (Williams 1969: 256). It operates language, and it works while you sleep. Yet, the concept of mechanical literature, and poetry in particular, is fascinating, and McHale (2000: 11)

tells why. Answering the question why computer-generated poetry '... reads like some mode or other of contemporary American avant-garde writing ...' he suggests that some of these avant-garde texts ' ... may themselves have been generated "mechanically" '. He proceeds to distinguish four methods of mechanical composition: imposing constraints, applying procedures, imposing selection and using combination – indeed the ingredients of modern construction grammar for natural language. Men's built-in text generator may behave mechanically now and then.

The basic hypothesis that we will investigate is that language entertains a non-trivial level of semantic computability, and that meaning is the tractable product of an (almost) mechanical architecture – meaning, not truth, that is, nor adequacy, information, normality or any other non-truth-functional epiphenomenon of language-in-use. Our claim is not that all meaningful aspects of language are computable; in particular, we don't believe that the contextual aspects of meaning are in the scope of deterministic algorithms. Pragmatics of Gricean breed may be understood as a level of meaningful analysis by Levinson (2000) and others: conversational implicatures *e.g.* assume intelligence, awareness and a theory of mind which we consider to reach beyond the minimal assessment for human language learning, usage and understanding. We talk about those aspects of meaning that are not affected by a human's degree of autism. On the other hand, no sentence varies its meaning over the infinitely many contexts in which it can be used. Those components of meaning that do not vary with contexts but rather stay constant are the ones we are after. For example, there are no contexts in which a sentence and its negation mean the same. Their pragmatic impact may be the same, however, for example in a situation where one just says something irrelevant to break a painful silence. Therefore, the difference between a sentence and its negation is a dimension of meaning that we want to explore by computation. This bottom line of meaning we consider to be '... the "holy grail" not only of linguistics, but also of philosophy, psychology and neuroscience – not to mention more distant domains such as cultural and literary theory' (Jackendoff 2002: 268).

Although the aspects of meaning that we aim to compute are rather elementary when compared to 'the pragmatic penumbra closely surrounding sentence-meaning' (Levinson 2000:1), analysis of meaning cannot do without these elements. Even for determining the relevance, adequacy and specificity of a text with or without conversational implicatures, all the small and inevitable seeds of the sentences in it have to be taken into consideration. The big meaning lives on small components, emanating from sentence construal. In

that sense, this book is about the grammatical conditions for computing the propositional molecules of meaning in its broadest sense.

Sentences are meaningful because most strings – statistically: almost all strings – of words are not well formed. To tell the fraction of well-formed and meaningful strings from the virtual overflow of verbal nonsense is the task of grammar, and of a grammar machine. We are not talking furiously sleeping green ideas here. Some sentences may sound weird *because* they are meaningful. The grammar cannot but approve of them, for exactly that reason. The grammar-in-action must identify that utterly small-though-infinite subset of serial words that happens to contain the meaningful propositions, whether they are or turn out to be weird, false, insulting or inadequate. A grammar machine must not stick with Boolean judgements. It arrives at a decision by computing structure, and this structure conveys the semantics. All we have to show, then, is that this semantics is not trivial. That amounts to the view that there is meaning to a proposition, and not just to words. For the present purpose, we will even assume that the meaning of words is *not* the non-trivial level of meaning we are after. The hypothetical level of analysis is the one at which semantic consequences can be established. That level is propositional, by nature. It is, just like syllogisms, not dependent on the particular meaning of words, but it assumes the meaningfulness of whatever constituent that plays a role.

Remarkably, the meaning of a sentence is neither trivial nor vague. Although this meaning is neither listed nor learned, it is almost beyond debate when compared to the meanings of words or to the pragmatics of a text. We need not agree on any particular word meaning in order to agree flawlessly on the meaning of sentences. But agreement, in this case, presupposes shared knowledge and shared interests. Consequently, our agreement on sentence meaning cannot stem from inspection of 'the world': that is exactly where our views and interest diverge. We can only – silently – agree on a domain we never debate: the structure of the language. Therefore, the non-trivial but unquestioned level of semantic computability cannot be established except by language structure. Propositional meaning is neither an accident nor an individual decision, but an intrinsic attribute of computed structure.

The general Turing machine models computability of the partial recursive functions. The Turing machine is neither a model of the human brain nor a model of computability in general. In particular, our brain may be able to compute stuff that is far beyond the reach of our Turing machines. When we try to 'do' language on a Turing device, we just try to determine whether or

not certain aspects of language are Turing-computable. The only importance of this game lies in the question *which* aspects of language can be shown to be Turing-computable. Here, we propose that propositional meaning and logical entailment are in this range. To the extent that we can establish this, it shows that an interesting part of the human language capacity is recursive enumerable or even more restricted. The part of the human language capacity we envisage here, is interesting because it contains semantic combinatorics. Given the computability of syntax and a compositionality ideology, one can argue that the computability of semantics comes for free. That is not true. The reason here is transparent: syntactic structure does not impose an upper limit to the number of propositions it induces. The possible propositions easily outnumber the given syntactic structures they may be related to. Linguistics, then, is bound to demonstrate that compositional semantics is computable all the way up and down. To hope, to say or to claim it is does not suffice. Linguists must do what Leibniz appreciated Llull for: calculate instead of debating the possibilities of calculation.

Consequently, this book is on linguistic engineering, rather than on linguistic theory. But then, linguistic theory is so incomplete or undirected that engineering has to fill the gaps. Another way to look at the problem involved is to say that theories, models and problem solving in linguistics are rather unbalanced. The theories may be *about* the data, and sometimes the data *are* the theory, as in naive constructionism. There is too little distance; there is so little distance that the theory can hardly guide the engineering. The point is made by Shieber (1988) and has hardly lost relevance. This book, therefore, presents linguistic engineering on all levels of sentence analysis, including the theory of grammar. In doing so, it assumes what we call *the language machine hypothesis:*

- that language is partly computable,
- that human language capacities are to be modelled by mechanisms rather than by representations,
- that human grammar is a process – a parsing or generating process – in the spirit of Marcus (1980), and that it is fast, faulty, correctable and goal oriented, rather than being guided by principles of efficiency, elegance and economy, and
- that an automaton may model but not describe the brain.

0.2 LANGUAGE AND COMPUTABILITY

Not every scholar will be convinced that the conjunction heading this chapter makes sense. Even in computational linguistic circles, the computability of natural language is disputed, in at least two senses. In one respect, some computational linguists believe that relevant parts of language are effectively beyond the type of formalization that is required for Turing-computability. In another though not unrelated respect, some computational linguists doubt whether such formalization can be done efficiently. This second type of objection is taken for granted here. In a certain sense, even the authors of this book are convinced that a whole class of language tasks can be performed more efficiently by statistical models than by recursive analysis. We do not believe, however, that interesting semantic tasks are in that class – notwithstanding the numerous efforts to perform inferential tasks by shallow processing, gathering in the TREC and other challenges; for a critique see Reckman (2009). We just try to show that effective computation of dynamic semantics on the basis of a lexicalized categorial grammar is within reach, for a non-trivial fragment of Dutch, on the parsing track as well as for generation.

For the basic construal here, the outline of which we owe to Jan van Leeuwen (p.c.), we are interested in the relation **Means** between sentences of Dutch (NL) and propositions from a formal language L. **Means(**φ, ψ**)** holds between $\varphi \in$ NL and $\psi \in$ L iff ψ is a meaning of φ. Furthermore, we will say that NL is semantically computable iff there is a partial recursive function *f* such that for all p in NL, if f(p) = ψ then **Means(**p, ψ**),** and L is enumerable. Clearly, this function *f* is a parser, sending p to ψ. Suppose we have that parser. Then, NL is semantically computable iff L is enumerable. If L has a finite lexicon – which we assume by definition – then its sentences are enumerable by induction on their length. For every q in this enumeration, check whether f(p) = q. There must be at least one, by the definition of *f*; the first one is found after a finite number of steps. Thus, NL is semantically computable if **Means(**φ, ψ**)** is decidable – equivalency is not at stake here.

The decidability of **Means** amounts to the general question whether human beings can agree on whether a sentence means something with finite means. In general, this is the anchor of successful communication, and there is little reason to doubt we can. On the other hand, L is a formal language, to which we don't have intuitive access. Therefore, we offer a formal way to test **Means**. Suppose we have the counterpart to *f:* a function *g* assigning NL sentences to

formulas in L. Now consider the relation **NLMeans(**φ, ψ**)** between sentences in NL. The relation holds if φ *means* ψ, that is, ψ can replace φ *salve veritate.* **NLMeans** expresses agreement between human beings on the semantic idem-potency of φ and ψ. We take it to be decidable: if humans cannot agree on semantic equivalence, who can? Next, suppose we have a reliable generator *g* mapping meanings onto sentences. Let *g* be such that for all propositions q, **Means(**φ, q**)** iff g(q, χ) and **NLMeans(**φ, χ**)** for φ, χ ∈ NL. In the presence of such a generating function *g,* **Means** is decidable if **NLMeans** is. QED.

In this book we try to partially define the functions *f* and *g* of the foregoing sketch of proof. The *f* is a parser assigning formal meanings to Dutch sentences and *g* a generator assigning Dutch sentences to formal meanings. To the extent that they turn out to be sound and reliable, the fragment of Dutch covered by them qualifies as a semantically computable language by the reasoning above.

The parser and the generator could have been designed as each other's inverse, but actually they are not. That is: in our approach, neither the parser nor the generator has an inverse, for functional reasons. The parser maps a sentence under a particular analysis to a family of meanings. These meanings are underspecified (cf. Bunt 2008): they do not – yet – specify the full net of interpretative dependencies that can be identified in a sentence. There is a post-derivational algorithm, inspecting the complete derivation of the sentence that 'spells out' the full inferential semantic network, including effects of scope and intensionality. The generator is fed with this type of input, but cannot invert the 'spell out' algorithm, for the simple reason that there is no one-to-one correspondence between fully specified meanings and sentences. In fact, both the parser and the generator map their input into classes of expressions. The parser maps a sentence in NL – Dutch – to a subset of L – the representation language – and the generator maps a proposition in L to a subset of NL. So, the parser *f* is a function from NL into the powerset of L, and the generator *g* is a function from L into the powerset of NL. Here is the picture.

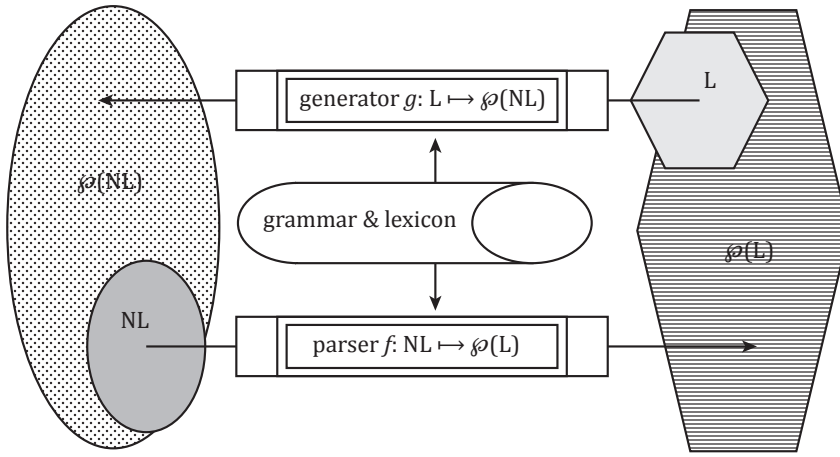

Figure 2 The parser and the generator as functions

Thus we claim that reversible grammars only exist outside the realm of fully specified meaning. The functions *f* and *g* cannot be inverted, as they target domains of a different order.

Talking about meaning, we can hardly escape the question what it is that language is about. Suppose that the interpretation of a language expression were to depend on and vary with the structure of its intended model. It is a good guess to assume that there are more than countably many possible models for a language expression. This interpretational dependency of a language expression on its model would, then, run into incomputability of meaning. Most certainly, however, the power to refer does not reside in language's capacity to refer to – or into or onto – a particular world. It is rather the other way round: language can refer because it does not presuppose any particular organization of what it is about. Dutch can be understood even if the world turns out to be created in six days and constitutes a prototypical black hole, inhabited by square circled demons. As a matter of fact, we talk in Dutch about every possible and impossible world. For language to refer, the minimal model is a non-empty set. That set provides denotations of type *e* – but they may be discarded in favour of generalized quantifiers, according to Montague (1972). The basic sets give rise, without any additional ontology, to subsets, providing denotations of type *<e,t>*, the predicates. One step further, we have sets of sets, of type *<<e,t>,t>*, the quantifiers. That is all we need. Reference to denotations of type *<<<e,t>,t>, t^n>* is obsolete, since those domains will not provide an essentially new algebraic structure (cf. Partee 1992). This three-layered window of denotational types – for names, predicates and quantifiers, basically – provides enough semantic structures to refer to whatever ontology – things, diseases, events, proposi-

tions, times, beliefs, ... This is what natural languages share with programming languages: the ontology is not fixed, but the syntax is. The difference between natural languages and programming languages is that the latter, but not the former, can switch ontologies in one single proposition, without declaration. This is what makes natural language universal and communicative, and the three-type window is an excellent candidate for *the* universal feature of natural language. For this window and its derivatives like functions from one type into another, the lambda calculus provides more than enough possible receipts to catch every possible denotation living there. Neither extensionally nor intensionally, will we easily run out of gas for our semantic machinery.

The basic concept behind Turing's and Church's notions of computability is recursion. The computable functions are the partially recursive ones: those the computation of which can be performed by a sequence or composition of functions with 'smaller arguments' or of less arity, up to those functions that are called primitive recursive. Recursion, more than anything else, is what links the computation of meaning in natural language to Turing computability. Unfortunately, recursion of semantic construal has been mixed up with compositionality in linguistic scenes: the meaning of a structure depends on the meaning of its parts. Some call it *Frege's principle* (see *e.g.* Heim and Kratzer 1998) but Janssen (1986) denies this attribution. *Compositionality* is an unfortunate term because it has been used to detract from the process: the meaning of the whole is a function of the meaning of the parts. Interpretation, though, is a process, not a property, and it is the recursion of the composition process, rather than the meaningfulness of the proper parts of a constituent, that must guarantee the semantic validity of the outcome. On the other hand, mathematics is the art of describing processes in a stative way. But still, what we are looking for is a *procedure* to apply meanings to each other, recursively. Semantic recursion and compositionality are not equivalent concepts, in this respect. A layered treatment of an oracle's proclamation of meaning may be recursive, but not compositional, and the simple one-step concatenation of meanings may be compositional but not recursive. This observation is crucial. The proposition assigned as a meaning to a sentence may itself be composed of propositions, but natural language is not such that simple concatenation suffices to assure that assignment. That is: if *abc* is a well-formed sentence meaning φ and the meaning of *a*, *b* and *c* are the conjunctions of propositions $p_{a1} \& \ldots p_{ai} \ldots \&\ldots p_{an}$, $p_{b1} \& \ldots p_{bi} \ldots \&\ldots p_{bm}$ and $p_{c1} \& \ldots p_{ci} \ldots \&\ldots p_{ck}$ respectively, then we know for sure that $\varphi \not\equiv p_{a1} \& \ldots p_{ai} \ldots \&\ldots p_{an} \& p_{b1} \& \ldots p_{bi} \ldots \&\ldots p_{bm} \& p_{c1} \ldots p_{ci} \ldots \&\ldots p_{ck}$. More generally:

(1) If ⟦abc⟧ = φ then $\varphi \not\equiv$ ⟦a⟧ & ⟦b⟧ & ⟦c⟧

The basic idea behind this lemma is that the meanings of the parts are supposed to be intertwined in a way that cannot be modelled by simple concatenation. In particular, we assume that the parts have variable-like placeholders that have to be synchronized. This is where Montague's Proper Treatment of Quantification in Ordinary English comes in (Montague 1972). Here meanings are composed by typed lambda conversion, in particular by function composition. The functions are such that the output of one is input to the other. As a consequence, variables are 'pipelined' and the resulting propositions become connected: every variable that was abstracted over is converted 'away' (or bound by operators outside its origin) in the composition process. Modelling this recursive process of composition is the challenge for computational semantics. The recursion must be steered in such a way that the place-holding variables are accounted for properly. Conjunction alone won't do. Function composition might. Still, we will argue in chapter 2 that the composed meaning can be designed as a conjunction of small clauses: unordered self-contained semantic objects in their own right. But this conjunction is neither immediate nor a trivial consequence of structure.

Semantic recursion stresses the arbitrariness of the primitive argument. The difference relates to the nature of semantic units. A complex phrase may be semantically atomic (or less complex) whereas its proper parts may be subject to syntactic manipulation. Picking up the phrase's meaning can be done recursively, but not very compositionally. Since phrasal meanings are utterly important – and, as constructionist linguists argue, dominant – in sentence construal, recursion is more adequate than composition. Yet, one of the most prominent approaches in computational semantics is dubbed *Minimal Recursion Semantics*, where *minimal* modifies *recursion* (Copestake *et al.* 2005). The basic idea, also native to the approach applied here (chapter 2), is that the derivation specifies a family of interpretations – a *packed forest* – and that the spelling out of full meanings is done post-derivationally, according to a protocol that 'reads' and unfolds the packed forest. The family of interpretations is dubbed *under-specification*. Essentially, unpacking the underspecified forest is not part of the grammatical combinatorics. Thus, the computation of the full meaning is set apart from grammatical computation, though fed by it. As a consequence, the nasty implications of spurious ambiguity for the complexity of grammar are avoided by sequencing algorithms, in a way comparable to the cataract of finite state mechanisms that has been developed to fight the opaqueness of monolithic systems (*e.g.* Van Noord 1997).

To make a long history short: from our perspective, Aristotle, Frege, Church, Chomsky, Montague, and Van Benthem equally contributed to the insight that language can be captured by formal means. Aristotle revealed the laws of syllogism and the algebra of reasoning with words. Frege detected the polarity of reference and meaning. Church made functions operational all the way down. Chomsky re-styled grammar, for linguistics and psychology. Montague gave meaning back to grammar. Van Benthem tied lambdas, types and derivations together. And then, there were generalized quantifiers (Barwise and Cooper 1981, Zwarts 1981) and they were the first semantic objects the algebraic structure of which turned out to be linguistically relevant: definable, testable, refutable creatures in an algebraic landscape, interfering with the distribution of phrases and connected to very general, if not universal, properties of natural language (Zwarts 1983). Their being there and their nature made clear that language lives somewhere in the brain, and that it makes itself known. There is no reason to underestimate or obscure or neglect it. There is a system that facilitates conveyance of thoughts, findings, emotions. It may have arisen from the lust for communication, or from the lust for expressing one's thought or from both – communication is the interaction between the verbalized straw men of our minds. Like most things in evolution, it is not designed to do what it turns out to do, but it is heavily affected by its function. That system is computable, partly by symbolic means according to Marcus (2003), and it may as well have a Dutch face. *Quod sit demonstrandum.*

0.3 THE BOOK

As we maintain a rather conservative view both on computational linguistics – it is linguistics – and on semantics – meaning exists and can be computed – the book has a rather conservative set-up. It consists of three chapters devoted to the main linguistic engines of the language machine: syntax, semantics and the lexicon. And it concludes with a fourth chapter in which the relationship between the three engines is reviewed.

The chapter on syntax introduces a rigid, modal, constructionist, combinatory categorial grammar, Categorial List Grammar (CLG). Its main linguistic and computational properties are made explicit and evaluated. The chapter describes the main syntactic patterns of Dutch and the way CLG covers them, in an all-and-only perspective, while linearizing strings and structuring com-

plex symbols at the same time. Furthermore, the chapter scrutinizes how the CLG is exploited in parsing and generation by the Delilah system.

The chapter on semantics presents three different but related instances of logical form, dubbed Stored, Applied and Flat LF. They differ in specificity and the way they are derived. The need for the threefold is established by reference to the variety of tasks with which a formal interpretation of natural language has to comply. In particular, the interference of compositionality with reading selection, scopal phenomena, intensionality and semantic inference is highlighted. The chapter describes how parsing in the Delilah system leads to a fully specified semantic representation and how this fully specified semantic representation can drive generation.

The third chapter deals with the data for meaning-driven parsing and generation, the lexicon. The Delilah system is lexicalistic, in an almost extreme way: it holds entries – lemmas, in effect – for each use of each phrase. The explosion of phrases following from this strategy is countered with smart indices and utterly fast retrieval. The lexicon consists of unique fully specified complex symbols, in the form of graphs open to unification. The chapter describes the off-line construction and the on-line retrieval of the database. The way we exploit the lexicon efficiently for parsing and for generation is revealed. As a matter of fact, the lexicon described here is more of a *constructicon*: it holds the phrasal components of Dutch in a non-hierarchical, matrix-like database, the front-office access to which is much more relevant for computability than its back-office genesis and hidden structure. In these chapters, essential data structures are taken from the Delilah operation mode. The make-up of the data structures may slightly differ depending on the process module handling them. Notwithstanding these differences, they always represent the very same type of underlying graph. In discussing the data structures we do not differentiate between categories and types in a fundamental way.

The final chapter proposes a revision of the formal relationship between syntax, semantics, and the lexicon. It claims that some of the less elegant solutions in the preceding chapters trace back to a fundamental incongruence in the architecture of formal grammar. It claims that grammar is bound to be incomplete – that certain valid statements about form and meaning cannot be derived by valid syntactic or semantic computations, and that this is a desirable state-of-affairs. In this sense, we feel this book to be a positive interpretation of Hugo Brandt Corstius' almost untranslatable 'first law of computational linguistics': *Wat men ook doet, de betekenis gooit roet* (Brandt Corstius 1978: 110) – 'Whatever you do, the semantics comes through'.

To summarize, the book is meant to be a defence of at least the following claims:

- the propositional content of a Dutch sentence can be computed;
- for every proposition, a Dutch sentence can be computed that entails that proposition;
- the proposition associated with a Dutch sentence can be represented as a flat conjunction of small clauses, indexing the propositional content, with skolemized events and states as a backbone;
- when computing meaning, underspecified and specified levels of representation are produced by essentially different algorithms;
- given this separation, no homomorphism between syntactic and semantic procedures needs to be established for computational reasons;
- the parser computing propositions and the generator producing sentences are not each other's inverses but exploit the same lexical and grammatical resources;
- underspecified levels of meaning can be computed by unification;
- the lexicon of Dutch is phrasal;
- the lexicon of Dutch can be organized and accessed as a non-hierarchical family of complex signs;
- all relevant unifications in the grammar of Dutch can be controlled by a limited number of modes defined on a rigid categorial grammar;
- formal grammar is incomplete iff it is consistent.