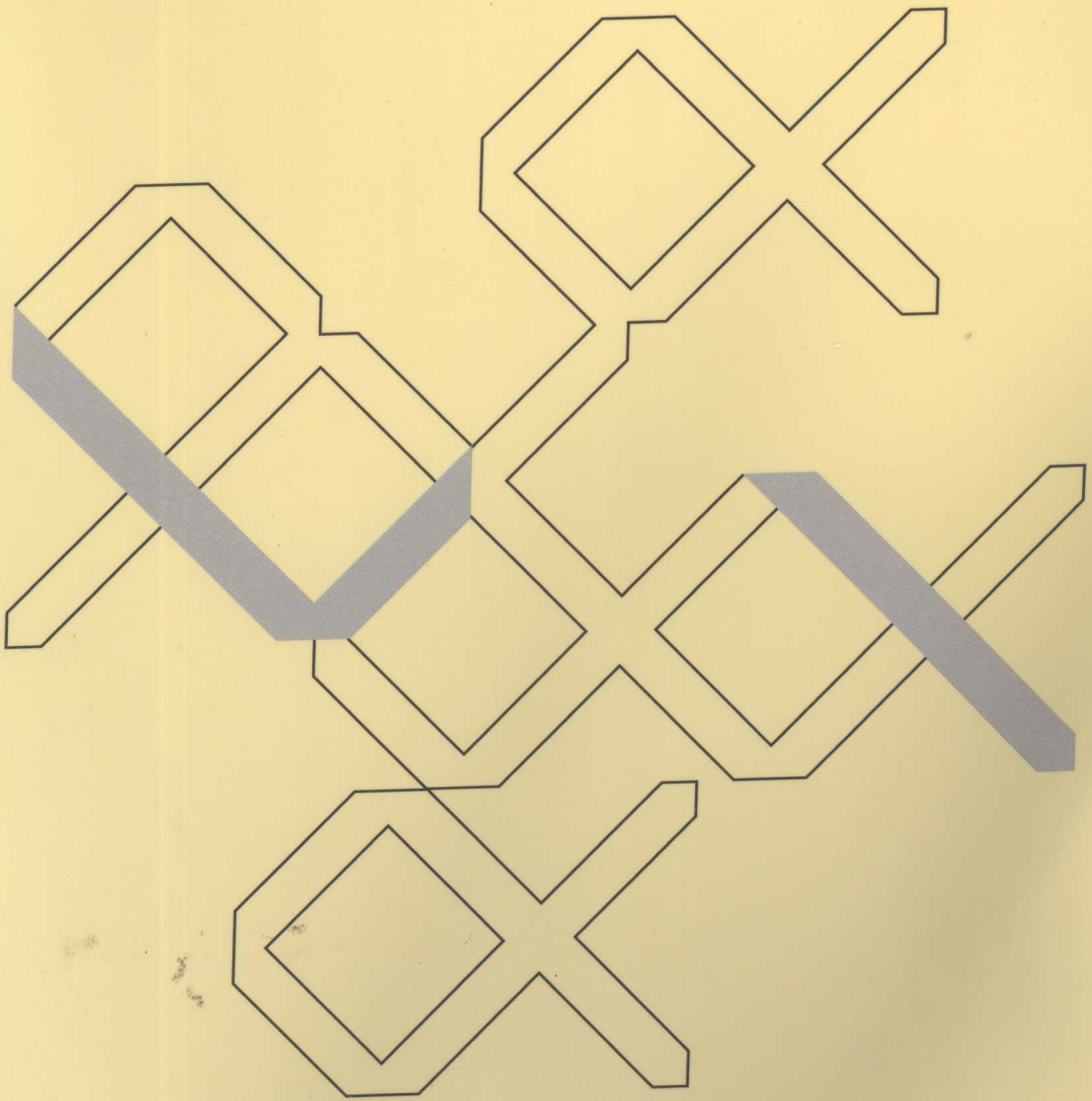
Linguistic Inquiry
Monograph Forty-five

Interface Strategies

Optimal and Costly Computations

Tanya Reinhart



Interface Strategies

Optimal and Costly Computations

Tanya Reinhart

The MIT Press
Cambridge, Massachusetts
London, England

MIT Press books may be purchased at special quantity discounts for business or sales promotional use For information, please e-mail special_sales@mitpress.mit.edu or write to Special Sales Department, The MIT Press, 55 Hayward Street, Cambridge, MA 02142.

This book was set in Times Roman and was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Reinhart, Tanya.
Interface strategies : optimal and costly computations / Tanya Reinhart.
p. cm.—(Linguistic inquiry monographs ; 45)
Includes bibliographical references and index.
ISBN 0-262-18250-5 (alk. paper); 0-262-68156-0 (pbk: alk. paper)
1. Grammar, Comparative and general. I. Title. II. Series.
P151.R45 2006
415—dc22 2005054004

10 9 8 7 6 5 4 3 2 1

Acknowledgments

Kriszta Szendrői has made important contributions to this work since 1999, when I presented my ideas on focus and stress in the summer linguistics school in Potsdam, Germany. Her dissertation (Szendrői 2001) has shaped much of my analysis of stress in chapter 3, and her research as a post doc in Utrecht since 2002 has turned the acquisition hypothesis entailed by the focus analysis into reality. It is hard to imagine that chapter 5 would have materialized without her ideas, experiments, and constant feedback. In this area of stress and focus, I also wish to thank Ad Neeleman, who taught me how important PF is, and how to think about it. Many of the arguments in chapter 3 were formed in our joint work (Neeleman and Reinhart 1998).

My views on economy, reference-set computation, and the economy approach to anaphora were formed in intensive interaction with Danny Fox since 1993, when we were both exploring Yael Golan's idea that reference-set comparisons are relative to interpretation. His inspiration goes beyond what is evident in the chapters on these topics.

Choice functions were conceived through endless conversations with Remko Scha in the early 1990s. His patience, knowledge, and wisdom were formative at the stage summarized in Reinhart 1992. At later stages, many of the ideas were developed further through collaboration with Yoad Winter. Eddy Ruys has been a constant source of scrutiny and feedback on all issues of QR, economy, syntax, and interpretation.

An acquisition conference in Trieste in 1998 provided key input for this work. Gennaro Chierchia, Stephen Crain, Maria Teresa Guasti, and Rosalind Thornton presented their early findings on the acquisition of scalar implicatures there. It was the first indication that 50 percent performance (chance) is found in another area of reference-set computation besides coreference. So what was by then just a hypothesis seemed to become an experimental reality. My exchanges since, particularly with

Stephen Crain, have been extremely valuable as I pursued this hypothesis further.

The final shape of this book owes a lot to anonymous reviewers, of this monograph as well as of previous articles that were partially incorporated here. Their thorough and insightful comments gave me even more food for thought than I could incorporate.

Contents

Introduction: Optimal Design

A hypothesis that got much attention in the 1990s is that the well-formedness of syntactic derivations is not always determined by absolute conditions, but it may be based on a selection of the optimal competitor out of a set of candidates—a reference set. A restricted version of this was assumed at the early stages of the minimalist program (Chomsky 1992, 1994), and simultaneously, it has been the central notion developed in Optimality Theory (OT) (see Prince and Smolensky 1993 and later work on optimality in syntax, including Grimshaw 1997). The computation of optimality selection is of a different sort than previously assumed in syntax. Typically, it requires that in computing a given derivation, an alternative derivation be constructed, in order to determine whether a given step in the current derivation (or the full output) is permitted. I will refer to this type of computation as *reference-set computation* (following the notation in the early minimalist framework). Naturally, the introduction of this new type of computation ignited a debate on whether the computational system of natural language (syntax) indeed includes such computations. In later developments of the minimalist framework (since Chomsky 1995), it was determined that there is no evidence that reference-set computation applies in core syntax. To establish that such computation is required it would be necessary to show that whatever it derives could not be derived otherwise, with more minimal computations. In fact, all original syntactic arguments in favor of this computation have found a simpler explanation in the minimalist framework.

The line of argument I pursue here is that this computation is nevertheless available to the computational system, as witnessed in some areas of the interface. But it is much more restricted than assumed in OT. It applies only as a "last resort," when the outputs of core syntax operations are insufficient for the interface. The instances of reference-set computation that I examine in subsequent chapters can be viewed as interface

strategies needed to make up for imperfections in the system. Their operation is severely restricted, and applying them comes with a processing cost.

The concept of the interface that underlies this work can be best illustrated with a thought experiment from Chomsky 2000—an "evolutionary fable." Imagine a primate that by some mystery of genetic development acquired the full set of the human cognitive abilities, except the language faculty. We can assume, then, that among other cognitive abilities, he has a system of concepts similar to that of humans, and a sensorimotor system that enables the perceiving and coding of information in sounds. Let us assume, further, that he has an innate system of logic, an abstract formal system, which contains an inventory of abstract symbols, connectives, functions, and definitions necessary for inference. What would he be able to do with these systems? Not much. Based on the rich concept system of humans, his inference system should in principle allow him to construct sophisticated theories and communicate them to his fellow primates. However, the inference system operates on propositions, not on concepts, so it is unusable for the primate in our thought experiment. Possibly he could code concepts in sounds, but not the propositions needed for inference.

Pursuing this thought experiment, the goal of linguistic theory can be described as reconstructing the system the primate lacks, which consists of whatever is needed to facilitate the interface of his various cognitive systems. In other words, the goal is to construct the computational system (CS) (syntax in a broad sense) that defines language (L), a state of the faculty-of-language (FL) organ, which makes this interface possible. Correctly capturing the interface is the crucial adequacy criterion of any syntactic theory. This is not to be confused with functional accounts of language. There is ample evidence by now that it is strictly impossible to derive the properties of the computational system from any functional considerations of language use. Systems of inference, use, and communication are consistent with many possible languages, and they cannot explain why the particular human language was selected. On the other hand, it is a crucial fact about human language that it can be used to argue, communicate, think, and so on. If our formal analysis of the computational system turns out to be inconsistent with basic facts of language use—for example, if it can be shown that the representations it generates are unusable for inference or cannot adjust to varying contexts of use—this cannot be the correct analysis, since the actual sentences of human language can be used for such purposes.

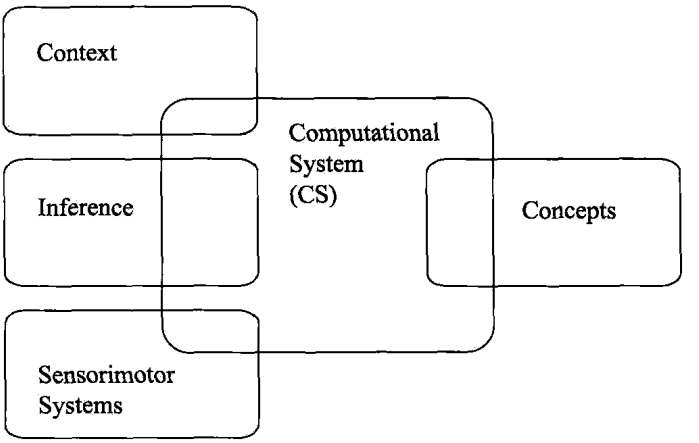


Figure I.1

Making the interface possible means that the other cognitive systems should be able to access the representations generated by the CS, namely, these representations should be legible to the other systems. Chomsky defines the "interface levels" as sets of representations legible to other systems (external to the faculty of language). He outlines two broad external systems (each consisting of sets of systems): (1) the sensorimotor, articulatory-perceptual systems, and (2) the conceptual-intentional (C/I), or thought systems. But for work on the interface it may be useful to decompose further the components of the C/I interface. I assume three (sets of) such systems, along with the sensorimotor sound system, as schematized in figure I.1.

We may assume that basic information of the concepts system is coded on the lexical items, which are the building blocks of the CS derivations. For this purpose, the information must be coded in a form legible to other systems—for example, as thematic features. The problem of legibility with the concepts system is somewhat different from the problem with the other C/I systems. Some of the information of the concepts systems, like the number of Theta roles of a verb—the verb's arity—must be legible to the CS (rather than conversely). Similarly, the thematic properties of a selected argument may determine its merging order in the derivation (agents must merge in SpecVP, and so on). Much of the other information coded on lexical items is not legible to the CS itself, but it is transferred through the derivations of the CS to the other C/I systems, and it should be legible to them. (This is the same intuition that underlies the concept of interpretable features in the minimalist framework.) The

inference system is essentially logic and its inventory includes, for example, logical relations, functions, abstract predicates, and variables (but no constants). The outputs of the CS are representations that are legible to the inference system, which can read them as propositions fit for its computations. The hardest to define given our present state of knowledge are the context systems that narrow the information transmitted through the derivation (coded in the relevant representation), and select the information that is useful for the context of use. On this view, then, the C/I systems are the concepts/context/inference systems. Figure I.1 is, of course, an abstraction. It may turn out to be necessary to assume that the context, inference, and sound systems may have direct interfaces, rather than each negotiating only with the CS, as in the figure. (This is related to Chomsky's (2000) question of association.) I will touch occasionally on this question, particularly in chapter 3, on focus and stress, but I do not attempt a comprehensive answer. The specific focus of this book is the interface of the CS with the inference and the context systems. The interface with the concepts system is the topic of research on the relations between the lexicon and the computational system. I discuss the concepts interface in Reinhart 2002.

A central question of linguistic theory, then, is how the interface is guaranteed, or what makes the CS representations legible to the other systems. Put more broadly, the question is how structure and use are related. There is no pretheoretical way to answer this. Suppose we observed, empirically, that a certain derivation D is associated with a set U of possible uses. In principle, there are several conceivable ways that this could come about. One is that the properties necessary for U are directly coded in D, through the computational system, as specific features, functional projections, operations, or conditions on derivations. In other instances, it is possible that there is no direct relation between the syntactic properties of D and U. Rather, the set U is determined solely by independent properties and computations of the external systems, which apply to legible CS representations and further modify them. Yet another possibility is that there are some interface strategies associating D and U, using independent properties of the CS, and of the external systems.

I believe all three solutions to the question of the interface are realized in various areas of the interface, but the one actually favored in syntactic practice is the first—that of syntactic coding. Many of the properties now encoded in the theory of the CS got there in order to guarantee the correct interface with the external systems of use. Quantified (Q), focus (F), and referential (R) features are just a few examples. The features

approach has been found useful in current syntax. The theoretical goal is that syntactic operations—the computational system—should be driven only by purely formal and mechanical considerations, and feature checking is a formal procedure of this nature. Nevertheless, it is far less obvious in advance that the full information necessary for the interface is coded in the CS in the same way. What it would mean, if true, is that features of one cognitive system are fully coded in another. It is easy to see why this is an attractive option. If this is how the human system has developed genetically, it can be viewed as a perfect system, where full matching of the various cognitive systems is guaranteed in advance, and no problem of coordination (or of the interface) can ever arise. Though not impossible conceptually, more evidence that language is perfect in this sense is needed than is currently available. We should also bear in mind that if the properties we encode in the CS (as theoreticians) do not, in fact, belong there, we are unlikely to get very far. As we will see, encoding interface properties in the CS has led to an enormous enrichment of the machinery. In many cases, the result is a highly baroque syntax, which, nevertheless, fares rather poorly in capturing the interface.

This book covers four areas of the inference/context interface: quantifier scope, focus, anaphora, and (more briefly) scalar implicatures. The first question in each of the areas is what makes the computational system legible to the other systems at the interface. In other words, how much of the information needed for inference and context is coded in the CS, and how is it coded? Once the CS coding is established, we discover that in each of these areas there are certain aspects of meaning, or the use of derivations at the interface, that cannot be coded in the CS formal language, on both conceptual and empirical grounds. This residue, I argue, is governed by interface strategies that can be viewed as strategies of repair, adjusting the derivation to the needs on the interface. (Not all interface strategies are strategies of repair, but those discussed here are.)

The broader context of this analysis is Chomsky's (2000) hypothesis of optimal design. The term *optimal* as used here should not be confused with its use in Optimality Theory (OT). In OT, optimality is a type of computation—selecting the optimal competitor out of a reference set. Here the question is whether the genetic design of language happens to be optimal.[1] As we saw with the primate thought experiment, the problem that language is a solution to is the interface of the different cognitive systems; to enable this, its representations must be legible to the other systems. Chomsky's working hypothesis, viewed as an ideal to guide inquiry, is that the solution is optimal: "Language is an optimal solution

to legibility conditions" (p. 96). A difficult question, of course, is what would count as an optimal design. A useful way to think about this may be to imagine a spectrum between a perfect and a poor solution, and optimal systems should be closer to the first alternative than to the second. As a first approximation, we may take the earlier view of the minimalist program, which I discuss in chapter 1. The assumption is that in a perfect system, the bare minimum needed for constructing derivations will be sufficient for the full needs of the interface. We may view deviations from the perfect system as imperfections—adjustments needed to enable the output representations to meet interface requirements. The less of these there are, the more optimal the system design is. Though this is not the full story, we may note that in the three interface areas under consideration here, the CS outputs are not sufficient for the interface needs. Some extension or repair of what the CS allows is therefore needed, so these are areas of imperfections. I will argue that the repair strategies involve the application of an illicit operation, which is only motivated by the fact that the output representations of the CS are not sufficient for the interface needs. Applying this operation requires constructing a reference set to check whether this is indeed necessary—that is, that this is the only way to meet the interface requirements. If optimal design has some measurable implications, imperfections should come at some cost.

While capturing the interface is the minimal requirement of the CS, a factor that cannot be ignored in determining which solutions can be viewed as optimal is that the actual use of language is also restricted by questions of "hardware." There is by now ample evidence that the human processor operates with limited resources of working memory and other limitations. So far we have looked only at the C/I interface. A CS that accommodates this interface would enable thought, but not yet communication. This also requires accommodating the SM (sensorimotor) interface, which, as formulated in Chomsky 2005, enables the externalization of language (sound, communication, processing). But the sensorimotor systems are severely constrained by the hardware available for sound production and perception, or more broadly, for the computations required in parsing sound inputs into linguistic representations. It may be possible to imagine various good solutions to the problem of the C/I interface, perhaps even better than the actual CSs, which require so much computation space that they would not be usable by humans. Suppose, then, that language is optimally designed to accommodate the C/I interface. The next question is how good the accommodation is to the SM interface, or the linking of the optimal design for the C/I interface with the fixed hard-

ware properties of the SM systems. Of the various hardware questions, the one relevant to the problems discussed in this book is how a parser that operates with limited memory resources can access and apply the definitions of the CS.

With this question in mind, we may turn to the fuller specification Chomsky (2000, 96) gives to the hypothesis of optimal design. He says: "Suppose that FL [Faculty of Language] satisfying legibility conditions in an optimal way satisfies all other empirical conditions too: acquisition, processing, neurology.... Then the language organ is a perfect solution to minimal design specifications. That is, a system that satisfies a very narrow subset of empirical conditions in an optimal way—those it must satisfy to be usable at all [i.e., the interface conditions]—turns out to satisfy all empirical conditions. Whatever is learned about other matters will not change the conclusions about FL." Note that what is outlined here is the (unrealistic) perfect solution (rather than the optimal). In the perfect solution, the CS is some kind of genetic development that, while optimally enabling the C/I interface, also happens to fit perfectly for actual use with limited resources (thus satisfying all empirical conditions, not just the subset needed for the C/I interface). Sticking to the question of the parser, along with Chomsky's reasoning in the paragraph quoted here, it is easy to see intuitively why this would be the ideal situation. The farther apart the CS and the parser are, the more questions arise regarding their coordination. For example, if a change takes place in one system, how does the other adjust? The question, then, is how close language design could be to this idealized perfect match. Phillips (1996) suggests that it is pretty close, arguing that "the parser is the grammar."

The question of how "transparent" the parser can be—to what extent it can directly apply computations of the CS, rather than its own independent algorithms—has a long history, with roots in Miller and Chomsky 1963. One interpretation of their proposal became known in research on processing as the Derivational Theory of Complexity. According to this theory, there should be a measurably greater processing load depending on the number and complexity of the operations assumed in syntactic theory (the theory even made the assumption that a passive sentence would impose greater processing load than an active sentence). The general contention since the mid-1970s was that this hypothesis did not find empirical support, and it was abandoned. Phillips reexamines this history in detail, and argues, first, that the empirical findings were not as sweepingly nonsupportive as they were said to be, and, more crucially, that this specific interpretation of Miller and Chomsky's hypothesis was not

necessarily warranted, because it also depends on the question of measurement, or perceptual complexity. It is not obvious that every difference in processing steps, or number of operations applied, should be measurable. Berwick and Weinberg (1984) argued that in a nonserial model of parsing, it is possible that increased complexity does not increase time demands.

Phillips's hypothesis that the parser is the CS adopts the strongest possible version of a transparent parser, which in Berwick and Weinberg's terms is *token-to-token transparency* between the CS and the parser. On this view, then, the fit of the parser and the CS is perfect.

But executing this hypothesis in Phillips's system also requires substantial changes in the CS, to make it usable in linear left-to-right parsing (changes that Phillips attempts to show are independently motivated by syntax-internal considerations, but that are nevertheless not consistent with current views of the CS). It may be realistic to consider the other option as well—that language is optimally designed, though it is not perfect. It has been repeatedly argued that in speech perception (processing) the parser uses specific principles or strategies that find no direct correlate in computations of the CS. The most famous are strategies resolving local ambiguity at a processing stage, which does not even arise in syntax, as in (1).

(1) a. Max knows Lucie well enough.
    b. Max knows Lucie will laugh.

In response time or eye-tracking experiments, it was found that there is more intense processing activity following the occurrence of *Lucie* in (1b) than in (1a). This indicates that the parser first attaches *Lucie* as a complement of the verb in both derivations, but then reanalysis is required in (1b). Such findings are often taken to suggest that the parser must be a system independent of the CS (other relevant parsing-specific operations will be mentioned shortly). But we may still ask how optimal the correlation between the two systems is.

Let us assume that the parser is some algorithmic device that generates trees. But it can lend itself to any other system for the specifics of the trees generated—that is, it has no internal information about what counts as a legitimate tree. This means that as long as the CS definitions and computations are accessible to parsing algorithms, the parser can construct trees defined by the CS as legitimate outputs. Though the parser can, in principle, parse anything that is formally compatible, it has developed to operate within the hardware of limited human working memory. Hence, there

are parser-specific strategies of how to minimize the load on working memory, while still applying the computations dictated by the system it works with, which in this case is the CS. A parser of this sort can be viewed as transparent, because apart from adjustments to hardware needs, it does not apply rules of its own, but borrows them from the CS—its computations are a function of the CS computations and the input string. (The term *transparent parser* has a long history, but I am using it only as described here.[2]) In practice, it may turn out that the actual human parser requires some parser-specific conditions for the processing of language derivations, but the more transparent the parser is, in this sense, the more optimal language design is. It means that the minimal design necessary to capture the interface also contains all the information needed for the parser; thus, the serious problems of coordinating two independent systems are avoided.

Perhaps the reason the idea of a transparent parser was abandoned was that for years it did not seem possible to define such a parser for the CS. The crucial problem seemed to be that parsing, unlike the CS derivations, proceeds from left to right (top to bottom). However, Pritchett's (1992) parser, which may not have received the attention it deserves, does solve this problem. It is a head-driven parser, which means that inputs are stored until a lexical head (verb) is reached. What distinguishes it from other head-driven parsers is a parsing condition (specific to language processing) stating that any step in the parsing derivation must satisfy a Theta requirement (Pritchett's Theta attachment). Once a parsing step is licensed by Theta attachment, other attachments in this parse follow the CS instructions. When the first verb is encountered, the subject being stored can be attached, by Theta attachment. The derivation proceeds bottom up, and at the same parse, VP, IP, and CP are constructed, the subject moves from SpecVP to SpecIP, and so on. (Let us abstract away from the question of adjuncts here.) In (1b), repeated in (2a), when *Lucie* is encountered, Theta attachment allows only one option—attaching it as the complement of the verb, just as in (1a). Hence, when this attachment is found inconsistent with the subsequent input, reanalysis is required.

(2) a. Max knew Lucie would laugh.
b. Max warned Lucie would laugh. (Garden path)

While (2a) is captured by virtually any of the available parsers without assuming either a head-driven or Theta-based parser, Pritchett's analysis focuses on the difference between (2a) and (2b). Both require reanalysis, but only the second is a garden path. Pritchett draws a crucial distinction