An Integrated Computational Approach to Binding Theory

Roberto Bonato

N° d'ordre : 3158

THÈSE

en cotutelle entre

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

et

L'UNIVERSITÉ de VÉRONE

DIPARTIMENTO DI INFORMATICA

présentée à

L'UNIVERSITÉ BORDEAUX I

par

Roberto Bonato

pour obtenir le grade de

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

An Integrated Computational Approach to Binding Theory

Soutenue le : 4 Mai 2006

Après avis des rapporteurs :

Claire Gardent .... Chargée de recherche, HDR
Eric Reuland ..... Professeur

Devant la commission d'examen composée de :

Denis Delfitto	Professeur	Examinateur
Claire Gardent	Chargée de recherche, HDR	Rapporteur
Roberto Giacobazzi	Professeur	Président
Christian Retoré ..	Professeur	Examinateur
Eric Reuland	Professeur	Rapporteur
Géraud Sénizergues	Professeur	Examinateur

– 2006 –

Advisors:
prof. Denis Delfitto
prof. Christian Retoré

Acknowledgements

First and foremost, I must thank professor Denis Delfitto. His constant encouragement, his patience, his intellectual enthusiasm and the ability to convey it to his students, his outstanding qualities as a scientist, a teacher, and a human being made my journey in academic research a truly valuable experience.

I thank professor Christian Retoré for having initiated me to computational linguistics, and for having given me the opportunity to start a PhD, and to join the "Signes" team that he created in 2002 at University of Bordeaux and that he leads with an original mix of scientific excellence and personal humor, gathering there a group of brilliant researchers around his interdisciplinary vision of what computational linguistics should be about.

I thank professor Giacobazzi for having encouraged me to hold on in tough times and, together with professor Masini as the coordinator for the PhD programme at University of Verona, having provided me with the best material conditions that a PhD student could ever dream of to carry on his work.

I thank my reviewers, professors Claire Gardent from LORIA, Nancy, and Eric Reuland from Utrecht University. Getting both praise and constructive criticism on my work from such outstanding world-class scientists has truly been an honor for me.

Thanks to all the friends who kept up during these years with my never ending whines about life, love, (lack of) money, research and assorted frustrations. Thanks to Kheira and Yannick, the dear friends of all seasons; to all the friends from the mystical land of Brittany, where I left my heart (or soul?); to Maxime, for having run to rescue me and my poor French; to Chiara and Laura, or "how to make linguistics reading groups charming"; to the good friends of Signes team in Bordeaux, for uncountable (bad) coffees taken together; to Paola, who matches in intelligence and sensitivity her husband.

Special thanks to Francesco, a good friend and a constant source of inspiration and high visions, for our life-changing eye-opening thought-provoking conversations late at night in front of a coffee machine or at some unspecified point of the cyberspace.

And thanks to my parents Ubaldina and Francesco, and my brother Alessandro, to whom I owe everything.

Contents

Preface

The core of this thesis mirrors my personal intellectual path, that of a computer scientist who tentatively makes his first steps into an unknown, hard yet fascinating discipline like linguistics. After having been initiated to the ingenuity and subtleties of formal semantics and Binding Theory, I tackled the somewhat mechanical task of encoding in the standard semantic framework for natural language the principles of traditional Chomskian Binding Theory of the early 80's. As the dissatisfaction with the resulting system grew, I delved deeper into the competing approaches that have been proposed since then. I discovered, in doing so, the extreme refinement attained by the intellectual tools of linguists' research and I tried to stretch my system in order to cope with their conclusions. In the end, with the invaluable guide of professors Denis Delfitto and Christian Retoré, I was able to transcend any ready-made solution that 30 years of linguistic research handed me to devise, standing on the shoulders of their gigantic insights, a unified computational framework which is, I believe, not devoid of interest neither for the linguist nor for the computer scientist.

I decided to preserve in my thesis the dynamic, trial-and-error structure of this never-ending process that is the true substance of what these years of academic research have taught me.

1

Introduction

Le langage est source de malentendus.

Antoine de Saint-Exupery, Le Petit Prince, 1943.

Computational linguistics (CL for short) is one of the most ancient domains of research in computer science, dating back to the first automatic translation projects of the early fifties. It seems like as soon as computers, powerful yet numb manipulators of symbols, were made available, they were used to tackle human language, the most complex and elusive symbolic system found in nature. Noam Chomsky, the founder of modern generative linguistics, is also known in the computer science community as the inventor of the immensely influential "Chomsky hierarchy" of the expressive power of (programming) languages. This is maybe the most famous example, but by no means the only one, of how generative linguistics and theoretical computer science are intermingled since their respective origins.

The term "computational linguistics" is somewhat ambiguous. In some contexts it is used to indicate the set of concepts and techniques for automatic processing of natural language: as such, it is considered a synonym for Natural Language Processing (NLP for short). This definition is far from being universally accepted. According to several researchers, CL stems from linguistics, with which it shares the eminently theoretical and explicative (as opposed to applicative) goal to unveil the principles and mechanisms that govern the human language faculty, conceived in CL as a purely *computational* process. This entails that concepts and techniques issued from computer science can be used to analyze it, such as logic, complexity theory, computability theory, or artificial languages theory. NLP can do away with explanations for how things actually are, as long as it devises algorithms that efficiently mimic human ability to accomplish different tasks on natural language texts, such as parsing, understanding, or extracting information.

The ambitious goal of the present work is to provide, by means of the enquiry on a specific problem which has both deep theoretical implications and important practical applications, an example of how it is possible to reconcile these two supposedly opposite perspectives.

1.1 Natural Language Understanding

During the last years, the need for language-based information retrieval and extraction technologies has provided an important impetus for the development of more and more sophisticated language processing systems. Most common applications (yet far from having attained full technological maturity) are spell checking, information retrieval, speech recognition, web-page processing, (un)supervised machine translation, automatic summarization or abstracting of technical texts, and spoken-language dialogue agents. From a scientific point of view, the most sophisticated tasks among them entail a deep level of linguistic knowledge and formalization of human language, therefore they give rise to several linguistic, mathematical, and computational challenges.

The formal approach to CL dominated the early stages of language processing research mainly due to the influence of the seminal work by Chomsky and others on formal language theory and generative syntax throughout the late fifties and early sixties (see [9], [15]), and the work of many linguists and computer scientists on parsing algorithms, initially top-down and bottom-up and then via dynamic programming. More recently, this field witnessed a shift towards probabilistic, knowledge-poor and data-driven methods: the availability of very large on-line corpora has enabled statistical models of language at every level, from phonetics to discourse. These approaches have been rewarded by relative success in specific domains like (un)supervised text classification, some sub-tasks of data-mining, and phonological analysis (for a comprehensive survey of statistical methods in natural language processing, see [41], [32]).

Nevertheless, formal approaches to computational linguistics are recently experiencing a renaissance due to multiple factors. In the field of logic applied to natural language, formalisms like Kamp's Discourse Representation Theory (or DRT, see [33]), Barwise and Perry's Situation Semantics (see [4]) report relative successes in dealing with traditional limitations of the seminal work by Richard Montague on the formalization of semantics for a fragment of a natural language (see [44], [43], [62]). The availability of more powerful computational resources makes possible practical implementations of algorithms stemming from formal approaches, traditionally more computationally demanding than statistical ones. Last but not least, intrinsic limitations of purely statistical approaches, which have been made even more evident by large scale technological developments, urge for a more fine-grained analysis of natural language texts.

It is our firm belief that human language is far too complex, structured, and subtle a thing to rely only on purely statistical methods to unveil all the information encoded in it. Although immensely successful and deservedly popular, in our opinion statistical and heuristics-based approaches to NLP are quickly reaching their theoretical ceiling[1], in particular when it comes to issues involving non-trivial semantic analysis of a text, like automatic summarization, natural language database queries, natural language generation.

[1] In aeronautics, this term refers to the highest altitude an aircraft can reach, which is usually a function of weight and power of the aircraft, and temperature and density of the air.

Future ground-breaking developments of natural language processing technology will have to deal in a non-trivial way with the elusive notion of the *meaning* of a sentence, or, more generally, of a text, be it to support an intelligent question-answering system, to generate grammatically correct and semantically meaningful sentences in natural language from an abstract representation of their meaning, to automatically synthesize an abstract of a technical text or to perform in-depth information extraction. Solving such tasks implies being able to correlate the syntactic structure and the meaning of a sentence in a systematic and computationally efficient way. Heuristic and statistical approaches are perfectly suited for tasks which do not involve a deep analysis of a text like automatic classification. But the subtle intricacies of meaning, over which even a small change in the position of words has potentially destructive effects, definitely lie out of their reach. Logicians, linguists, and semanticists have always acknowledged and wrestled with such immense complexity. It is time for NLP technologies to take advantage of the wealth of their work.

1.2 What this thesis is about

Let's consider the following sentence:

(1) After the woman who met John yesterday saw him, she told Bill that she was sure he would have trusted him like he trusted himself.

In every human language there are linguistic elements whose semantic content is not mediated by other linguistic elements. In (1) it is the case of *John*, *Bill*, *the woman who met John*. In every human language there are also other elements (usually referred to as "anaphoric") whose semantic content entirely depends on other linguistic elements. In (1) it is the case of *she*, *him*, *he*, *himself*. The computational problem posed by such elements is to find the (most likely) linguistic element they draw their semantic content from, usually referred to as their *antecedent*. This is a tough task, if nothing else because in general it involves several different modules of human language faculty: syntax, semantics, pragmatics, world knowledge and commonsense. Still, it is something that a human speaker can usually accomplish very quickly and reliably, on the basis of seemingly very precise rules. This problem raises two questions:

- What are the inner computational mechanisms that rule human ability to correctly compute antecedents of anaphoric elements in a sentence?
- How is it possible to mimic reasonably well such a faculty by means of efficient algorithms that work on real world natural language texts?

The view that we pursue in this thesis is that the answers to these two questions are not completely unrelated, and that in chasing a solution to one, we might stumble upon very good clues about the solution to the other.

1.2.1 The anaphora resolution problem

This thesis deals with a particular instance of the syntax-semantics interface problem, commonly known as the *anaphora resolution problem*.

There is little agreement in the linguistics community on the precise definition of what anaphora are (sometimes we'll use the English terms "anaphor" and "anaphors" instead of this Latin plural word). We will stick to an operational point of view in simply calling "anaphora" any situation in which a linguistic object gets its meaning or *denotation* from another linguistic object previously introduced. The object which "points back" is called an *anaphor*, and the entity to which it refers is its *antecedent*. The process of determining the antecedent of an anaphor is called *anaphora resolution*. So, in the following example:

(2) After the maid had killed the lord, she left the house

the pronoun *she* is the anaphor, and *the maid* is the antecedent. It is worth underlining the fact that the antecedent is not the noun *maid* but the phrase *the maid*.

We distinguish *intrasentential anaphors* (referring to an antecedent which is in the same sentence as the anaphor) from *intersentential anaphors* (referring to an antecedent which is in a different sentence from that of the anaphor). The focus of this thesis will be on the former, as a linguistic phenomenon which in our opinion lies at the very core of the interface between syntax and semantics. Furthermore, a sound and robust approach to intrasentential anaphora is the stepping stone on the way to a solution to the intersentential anaphora resolution problem.

Pronouns are omnipresent in every natural language text. Therefore, anaphora resolution should be a key task in machine translation, text summarization, information extraction, question answering, to name only a few NLP applicative domains. The approaches developed - traditional (from the purely syntactic ones to the highly semantic and pragmatic ones), alternative (statistic, uncertainty-reasoning etc.) or knowledge-poor - offer only approximate solutions.

After considerable initial research followed by years of relative silence in the early 80s, anaphora resolution has attracted attention of many researchers during the last 10 years and a great deal of successful work on the topic has been carried out. Discourse-oriented theories and formalisms such as Discourse Representation Theory and Centering Theory inspired new research on the computational treatment of cross-sentential anaphora. The drive toward corpus-based robust NLP solutions further stimulated interest in alternative and/or data-enriched approaches. Application-driven research in areas like automatic abstracting, data-mining, and information extraction, independently highlighted the importance of anaphora and coreference resolution, boosting research in this area.

1.2.2 Binding Theory

Binding Theory is a fascinating and controversial domain of theoretical linguistics that deals with the distribution and the denotation of a specific class of linguistic elements called Determiner Phrases or DPs (to which pronoun belongs, among others) occurring in a sentence. From a theoretical point of view, recovering the

semantic content of a pronoun appears as a problem that has to do with the meaning, i.e. *semantics* of human language; yet the distribution of pronouns in a sentence, as well as the relationships between their denotations seems to be subject to constraints which are essentially structural, or *syntactic* in nature. Binding Theory lies at the very core of the debate on the syntax/semantics interface in human languages.

Binding Theory was first formulated as a module of Government and Binding Theory by Chomsky in [12], where he stated three syntactic principles that govern the distribution of DPs in a sentence. Since then alternative interpretations to the same empirical evidence have been proposed, as well as to the principles that rule human judgements on this matter. The alternative approaches that we're going to take into consideration in the present work are Reinhart [50] and Reinhart and Reuland [54]. They have highlighted the importance of keeping different levels of linguistic competence clearly separated: what were formerly considered as syntactic phenomena are sometimes identified as intrinsically semantic, or delegated to the discourse theory level of description of human language. It seems like there's a trend in linguistic research on this subject that moves the burden of judgments on distribution of anaphors from syntax to semantics. At the far semantic side of the spectrum lies the very recent proposal by Philippe Schlenker [58], with which we compare our approach in the final chapter.

1.3 The idea: an integrated view of Binding Theory

This thesis presents a computational treatment of Binding Theory which integrates insights drawn from three among its most influential interpretations, proposed throughout last 30 years of linguistic inquiry.

Our goal is to integrate into the current framework of computational semantics the principles of Binding Theory. Research on how to systematically compute semantic representations of a sentence in first-order predicate logic, stemmed from the pioneering work of Richard Montague of the early 70's, has reached a considerable level of sophistication. Our starting points are the formalisms and procedures to compute in a bottom-up inductive fashion the predicate-argument structure of a sentence on the basis of its syntactic analysis. We aim at enriching these algorithms to incorporate in the output semantic representations the constraints over Determiner Phrases induced by the principles of Binding Theory.

Our enquiry starts from the classical Chomskian formulation of the principles of Binding Theory in the early 80's (section 4.3). We implement the additional apparatus needed to inductively compute and encode in the resulting semantic representation the mandatory/forbidden coreferential relations induced by the principles. The cumbersome notation and the computational shortcomings of the direct implementation of this approach are overcome by adopting the approach proposed later in the 80's by Tanya Reinhart (section 4.4). However, the direct implementation of this approach runs into some computationally intractable problems.

These problems are overcome in section 4.5, where we integrate into our system the basic insights of Reinhart and Reuland's "Reflexivity" interpretation. We operate an original synthesis of the previous approaches, moving from the simple

computational implementation of ready-made models issued from linguistics towards an original computational treatment which subsumes their basic stipulations under general computational principles. The result is a computational treatment of Binding Theory which is both effective and linguistically well-grounded.

1.4 Structure of the thesis

This thesis is structured as follows. In Chapter 1 we introduce anaphora (or coreference) resolution. This problem has both considerable applicative interest, and deep theoretical implications. As an applicative task, it is the object of active research (mostly by means of heuristic or statistical approaches) in the domain of Natural Language Understanding, in particular to achieve in-depth semantic analysis both of single sentences and of larger texts. From a theoretical point of view, this problem falls under the scope of Binding Theory, a whole branch of linguistics devoted to identify the principles and to unravel the mechanisms of human language faculty that rule the distribution and the denotation of Determiner Phrases in a sentence.

In Chapter 2 we provide the basic notions of predicative (as opposed to lexical) semantics of natural language which are necessary to understand the problem we want to tackle and set it in the context of modern research in computational semantics. We keep the syntactic and the semantic assumptions to a minimum in order to achieve as much generality as possible in our approach.

Chapter 3 shortly presents three among the most influential approaches to Binding Theory that have been proposed during last 30 years of research in generative linguistics. We also present the few cases of fruitful interaction of Binding Theory with Natural Language Processing to devise algorithms for coreference resolution, and with Computational Linguistics to address more theoretical issues like computational complexity characterization of human language faculty.

Chapter 4 is the core of our research. The first step is a simple implementation of the principles of traditional Binding Theory as it was first formulated by Chomsky in the early 80's. Then we tackle the problem from a different perspective, one inspired by the interpretation of Binding Theory first proposed by Tanya Reinhart in 1983. Both approaches present severe shortcomings that are overcome only through an original synthesis of the two previous approaches with Reinhart and Reuland's 1993 Reflexivity approach. The result is an integrated computational treatment of Binding Theory which combines insights drawn by two among the most influential approaches and subsumes them by means of general computational principles.

We devote Chapter 5 to the presentation of a very recent proposal of semantic treatment of Binding Theory due to Philippe Schlenker. Although conceptually very distant, our approaches come to comparable conclusions on several issues, and we highlight advantages and shortcomings of each approach. We prove that our system, like his, includes with no further assumption the principle of Denotational Economy.

2

Computational Semantics Basics

"Hi" he said, lying.

Robert Maxwell

The whole objective of computational semantics might be summarized as the answer to the following questions (quoted from [5]): what are the computational processes involved in semantic construction? And how can we automate the process of associating semantic representations with natural language expressions?

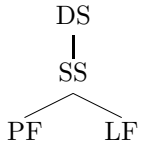
Computational semantics is concerned with computing the meaning of linguistic objects such as sentences, text fragments, or even (parts of) dialogues. It is the interdisciplinary field where semantics, the study of *meaning and its linguistic encoding*, meets computational linguistics, the discipline that is concerned with *computation on linguistic objects*. On one hand, it inherits concepts and techniques that have been developed in the domain of formal semantics, the linguistic discipline that applies the methods of logic to the description of meaning. On the other hand, this young discipline owes to computational linguistics the methods and techniques for parsing sentences of a natural language, for the effective and efficient representation of syntactic structures and logical forms, and for reasoning with semantic information.

The beginnings of computational semantics can be traced back to the seminal work of Richard Montague in formal semantics (see [44], [43] as well as [21] for an updated introduction). His work can be considered as the first accomplished effort to describe how the expressions of (a small fragment of) a natural language can be associated with semantic representations in a logical language. The focus of Montague's semantic theory is on the relationship between predicates and arguments in a sentence, or between categories and modifiers, and those set by variables bound by quantifiers. Such problems, after more than thirty years of active study in this domain, still lie at the very heart of any sound theory for natural language semantics, and are still a domain of active debate.

In the following sections we shortly recall the basic concepts and terminology for the computational semantic framework within which we will develop our enquiry on the computational treatment of Binding Theory of chapters 3 and 4.

2.1 From Surface Structure to Logical Forms

The generative tradition in linguistics that was initiated by the pioneering works of Noam Chomsky in the 50's (see [9], [15]) proposed a model for the relationships between syntax and semantics of human languages that is summarized in a schema which has become an icon of generative linguistics:



DS and SS stand for "deep" and "surface" levels of syntactic representation and can be thought of what we commonly refer to as *the syntax*: the set of rules that allow a speaker to perceive a sentence in his/her own language as well-formed. PF (phonological form) is a representation of the phonetic form of an expression; LF (logical form) is a syntactic level of representation which functions as the input to the semantic interpretation module. The three syntactic levels of DS, SS and LF are related by so-called *transformations*. Transformations are movement operations which move material from one position to another in the syntactic structure of the sentence. The transformations from DS to SS take care of word order relations between active and passive sentences, affirmative and interrogative sentences, etc. SS corresponds to a level of representation that is more or less a syntactically labelled version of what we see when we see a sentence. SS is the input for both Phonological Form (PF) and Logical Form (LF), which implies that it maps both on the actual phonological realization and on the input level to semantic interpretation. The split between PF and LF suggests that any syntactic movement which relates SS to LF is invisible to PF and as such is not phonetically realized.

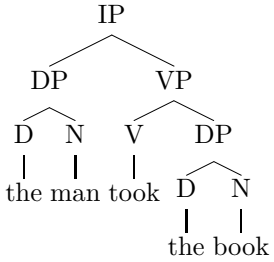
In the present work we are concerned with the SS and the LF levels of linguistic representation, and on the mutual relationship between them. Syntactic objects at SS will be the input data for algorithms which compute suitable semantic representations for them at LF.

2.1.1 SS input: phrase-markers

We shortly sketch the basic assumptions on the syntactic representations of sentences that will be used as the input to the semantic interpretation module. We inherit concepts and terminology on syntax issued from the tradition of generative linguistics initiated by Chomsky according to its latest formulation in the minimalist framework (see [14]).

Every word in the language belongs to a restricted set of grammatical categories, classes of expressions which share a common set of grammatical properties. Phrases and sentences are formed by successive applications of *merger* operations: two categories are merged together to form a new (phrasal) category. The resulting structures can be represented in the form of tree diagrams, usually referred to as *phrase-markers* or *parse trees* (we will use the two terms indifferently). Each of their nodes represents a *constituent*, which is the basic syntactic unit out of which the sentence is built. In this sense the tree representation of a phrase-marker

provides a visual representation of the constituent structure of the corresponding sentence. In the standard representation nodes carry labels which specify the grammatical category of the corresponding constituent. Every constituent has a *head*, which is the key word whose nature determines the properties of the overall phrase). There are as many different constituents in any given phrase or sentence structure as there are nodes carrying labels. By convention every constituent whose head is a word of category X is labeled as XP. Every pair of nodes belonging to the same phrase-marker is related by one of two different types of relation, either *dominance* (i.e. the hierarchical ordering) or *precedence* (i.e. linear, left-to-right ordering). What follows is an example of the syntactic structures that will be used as input to our semantic computations (possibly stripped from the labels that decorate the inner nodes[1]):



A common assumption is that the merger operation which leads to the formation of syntactic structures operates in a pairwise fashion to form larger categories out of simpler constituents. The immediate consequence is that phrase-markers in the present work will be always *binary* trees. Although not uncontroversial, such an assumption is widely accepted by the linguistics community and represents one of the tenets of latest formulations of generative linguistics. Hereby, all the phrase-markers occurring in the present work will be represented as binary branching trees.

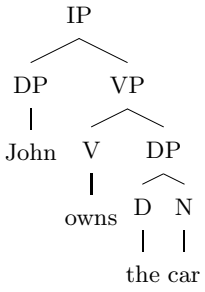
Our work will be mainly concerned with a particular kind of syntactic constituents called Determiner Phrases (or DPs). Intuitively, they can be thought of as the set of linguistic expressions that identify an *entity* in the domain of discourse: examples are *the king of France*, *John*, *he*, *him*, *the woman who married John*, etc. Binding Theory deals with the distribution and the mutual structural/denotational relationships between DPs occurring in a sentence.

2.1.2 LF output: First Order Predicate Logic formulas

The focus of our work lies in devising algorithmic procedures to automatically translate SS structures into suitable LF structures. LF is the syntactic level of representation of a sentence that directly interfaces with the semantic interpretation module. This means that at LF nothing is stated about the true *meaning* of the words involved, which is in general heavily context-dependent. What is commonly referred to as the Logical Form of a sentence, which is the target semantic representation we want to compute, is the representation of the *context-independent*

[1] Label D stands for Determiner, N for Noun, V for Verb, DP for Determiner Phrase, VP for Verb Phrase and IP for Inflectional Phrase, which is the category of sentences.

Table 2.1. Logical Form as intermediate representation between syntax and KR systems

Syntactic Analysis	Logical Form	Knowledge Representation
	$\exists!x : \text{CAR}(x).\text{OWN}(\text{JOHN}, x)$	$\text{OWN}(John\_Smith, CA073)$

component of meaning. In the present work the logical form of a sentence basically encodes the semantic relationships between words and phrases, that is their *predicate-argument structure*. However partial, this component of meaning is absolutely essential both for theoretical and applicative issues. Most of the research into formal semantics exclusively concentrate on this component of meaning, while any sound real-world language understanding system must be able to compute at least parts of it for fragments of a human language. Such kind of structure can then be fed to a contextual interpretation module which combines it with information about the specific discourse domain and generates a suitable representation into some Knowledge Representation language as illustrated by table 2.1.

The target language into which phrase-markers are going to be translated will be First Order Predicate Calculus. It provides a flexible, well-understood and computationally tractable approach to the representation of knowledge for a meaning representation language and a sound computational basis for the verifiability, inference and expressiveness requirements that are usually associated with the most common real world tasks for computational semantics. All these features make first order predicate logic formulas a reasonable and widespread choice for the semantic representation of context-independent meaning of a sentence.

2.2 Formal Semantics Basics

2.2.1 Truth-Conditional Semantics

What is commonly referred to as "formal semantics" for natural language can be simply seen as a way to associate predicate calculus formulas to a sentence, on the basis of its syntactic analysis. In the tradition of what is known as "the Fregean program" (from the logician Gottlob Frege, whose work in the late nineteenth century marked the beginning of both symbolic logic and formal semantics of natural language), we adhere to the notion of *truth-conditional semantics*: knowing the meaning of a sentence means knowing the conditions which make that sentence true or false in some world. This is not the only possible approach to semantics for natural language. To name just an example of an alternative view, *representational*

semantics characterizes the meaning of a sentence as a translation into some kind of internal mental representation of the world, and so it focusses on defining a translation relationship between utterances and a putatively universal system of mental representation (more on this philosophical debate can be found in [8]). In a truth-conditional semantics perspective we will say, for example, that the Italian sentence *c'è una balena rosa nel parcheggio dell'Università* is true if and only if there is a pink whale in the parking lot of the University. The apparent banality of such kind of claims hides a fundamental property that any formal theory of meaning for human language must be able to capture: namely, that we understand sentences (that is, we can say which set of conditions make those sentences true in real world) that we have never heard before (like - most probably - the sentence just quoted) out of the meaning of their parts on the basis of systematic computational mechanisms.

2.2.2 Compositionality

If we adopt a truth-conditional perspective, we can say that human beings are able to compute truth-conditions for sentences from the truth-conditions of their parts. Every meaningful part of a sentence contributes *in a systematic way* to the conditions that make that sentence true or false in real world. We adopt Partee's latest formulation in [47] of what is known as

Principle of Compositionality. The meaning of a compound expression is a function of the meanings of its parts and of the syntactic rule by which they are combined.

Compositionality principle lies at the very heart of the relation between syntax and semantics. It is also the trait-d'union between natural language semantics and computational semantics, and one of the basic assumptions of the project of formalizing the notion of meaning for (fragments of) natural languages.

Frege made a strong claim on the basic mechanism of semantic composition: he conjectured that it may always consist in the "saturation" of an unsaturated meaning component. As we will made clear in next section, this boils down to stating that the only mechanism by which syntactic constituents semantically interact with each other is functional application. This claim is known as *Frege's conjecture*, and even though modern formal semantics has departed quite significantly from it in dealing with some semantic phenomena, it still remains one of the leading ideas in computational semantics. Montague instead applied the compositionality principle on a rule-to-rule basis, assuming that for each syntactic rule, specifying how an expression can be built from simpler ones, the grammar contains a corresponding semantic rule that says how the meaning of the expression depends on the meaning of the parts.

It should be noted that Compositionality in its most general formulation does not necessarily require a rule-to-rule correspondence between syntax and semantics. The notion of *part* occurring in the principle is often understood as *constituent* in the sense of a substructure that has a significance in a syntactic structural description in the sense specified in section 2.1.1, but this is often an unnecessary

restricted interpretation. A grammar may define the set of well-formed expressions of a language by means of derivation rules without attributing a structural syntactic significance to the elements that are used in the rules.

Such a view is coherent with the actual trend in syntactic theories, both from a linguistic and a formal point of view, towards *lexicalization*. This means that the grammar of a language is completely defined by a lexicon which fully specifies the behaviour of words (terminal symbols in the formal sense), while derivation (or composition) rules are the same for any grammar. This idea is already present both in the linguistic community (for example in Chomsky's Minimalist Program [14], latest reformulation of his immensely influential work on generative grammars), and in the language theory area, where lexicalized grammar formalisms like Categorial Grammars (especially in their multimodal or Lambek variants, see [45]), Combinatory Categorial Grammars (or CCGs, see [61]), Tree Adjoining Grammars (or TAGs, see [31]) and Minimalist Grammars (or MGs, see [60], [40]) are the subject of intense theoretical research and promising practical applications.

2.2.3 Denotations and Types

Semantics for a fragment of a human language consists of three components: an inventory of denotations, a lexicon, and a set of composition rules.

Denotations can be things in real world, or digital representations of entities in a knowledge representation system, in general entities belonging to a certain domain of interest. Let D be the set of all entities that exist in the real world. Possible denotations are: elements of D, that is the set of actual *individuals* (that we use here as a synonym for *entities*); elements of $\{0, 1\}$, the set of truth values; functions from D to $\{0, 1\}$; functions from D to functions from D to $\{0, 1\}$, etc. Linguistic expressions are associated to such non-linguistic entities by means of an *interpretation function* $[\![\ ]\!]$, which assigns an appropriate meaning or *denotation* (from this point onward we will consider the two words as synonyms) to every syntactic object. We say that α *denotes* $[\![\alpha]\!]$ or, equivalently, that $[\![\alpha]\!]$ *is the denotation of* α. Examples (1) shows some examples of denotations associated to linguistic expressions by function $[\![\ ]\!]$:

(1) a. $[\![Mary]\!]$ = the real Mary in flesh and blood

b. $[\![sleeps]\!] = \{x \mid x \text{ sleeps } \}$

c. $[\![student]\!] = \{x \mid x \text{ is a student } \}$

d. $[\![Mary\ sleeps]\!] = \begin{cases} 1 \text{ if Mary sleeps} \\ 0 \text{ otherwise} \end{cases}$

In any compositional theory of meaning we want to be able to compute denotations of complex expressions from denotations of their simpler parts: for example, we want to describe rules to compute the denotation of (1-d) from the denotation of (1-a) and of (1-b). So, the interpretation function $[\![\ ]\!]$ must consist of two components: (i) a *lexicon* which associates basic meanings to words (terminal nodes in our syntactic trees) and (ii) a set of *composition rules* that describe how to derive the meaning of compound expressions (non-terminal nodes) from the meanings of their more basic components. We will get back to composition rules in section

2.2.4, while in the rest of the present one we will examine more carefully the formal machinery needed to associate denotations to words in a lexicon.

Example (1-a) shows that we assume that proper names denote individuals, (1-d) that sentences denotes truth values, (1-b) and (1-c) that intransitive verbs and nouns denote sets of individuals, and so their denotations can be identified with the corresponding characteristic functions from set of individuals to truth values. Those are *denotation domains* for, respectively, proper names, sentences, intransitive verbs and common nouns. Just like expressions belong to a syntactic category, they belong to a *semantic type*, according to what kind of denotation domains they denote in. We will call D_e the denotation domain for entities, individuals and other objects, and we will say that the semantic type of a proper name is e (reminiscent of "entity"). The denotation domain of sentences is the set of truth values $\{0, 1\}$, and the corresponding semantic type is t (which stands for "truth value"). But whatever language is, it is much more than just entities and truth values. Recursive definitions 2.1 and 2.2 supply us with as many semantic types, and their respective domains, as needed:

Definition 2.1. *The set $\mathcal{T}$ of semantic types is the smallest set such that:*

(i) e and t are types;
(ii) if τ_1 and τ_2 are types, then $\tau_1 \rightarrow \tau_2$ is a type.

If α is an expression of type $\tau_1 \rightarrow \tau_2$ and β an expression of type τ_1, then the result $\alpha(\beta)$ of *functional application* of α to β will be an expression of type τ_2.

Definition 2.2. *Denotation (or interpretation) domains for expressions whose types belong to $\mathcal{T}$ are defined as follows:*

(i) D_e, the set of entities or individuals, is the interpretation domain of type e;
(ii) $D_t = \{0, 1\}$ is the interpretation domain of type t;
(iii) for any complex type $\tau_1 \rightarrow \tau_2$, its interpretation domain is $D_{\tau_1 \rightarrow \tau_2} = D_{\tau_2}^{\tau_1}$, that is, the set of all functions from D_{τ_1} to D_{τ_2}.

For expressions of complex types $\tau_1 \rightarrow \tau_2$ we also say that they map elements from D_{τ_1} onto an element of D_{τ_2}. A proper name like (1-a) denotes an individual, and so it is of basic type e. The denotation of a sentence is a truth-value and so it is of semantic type t. Intransitive verbs like (1-b) or a common noun like (1-c) are of type $e \rightarrow t$, i.e. they map an individual onto a truth value, and so they are functions from individuals to truth values. The set of individuals *characterized* by such a function is the set of all individuals that the function maps to 1. Table 2.2 provides some examples of the semantic type (and thus implicitly of the denotation domains) of some natural language expressions.

Lambda (λ) calculus provides a useful notation to express the relation between linguistic expressions and their denotations in a compact way. Generally, λ-terms are constructed according to the following schema:

$$\lambda\alpha : \gamma \ . \ \phi$$

We say that α is the *argument variable*, γ *the domain condition* and ϕ the *value description*. The domain condition defines the domain of our function, and it does so by placing a condition on possible values of α. In particular, we adopt the following convention explicitly stated in [26]:

Table 2.2. Types and Expressions

Type	Kind of expression	Examples
e	Individual expression	John, the dog, the king of France
$e \to t$	One-place first order predicate	walks, loves Mary, student, dog
t	Sentence	John walks, John loves Mary
$t \to t$	Sentential modifier	not
$(e \to t) \to (e \to t)$	Predicate modifier	quickly, beautifully
$e \to e \to t$	Two-place first-order relation	loves, looks
$(e \to t) \to t$	One-place second-order predicate	every student, no woman
$(e \to t) \to (e \to t) \to t$	Two-place second-order relation	no, every, all

Definition 2.3 (λ-convention). *We read $\lambda\alpha : \gamma.\phi$ as*

(i) the function that maps every α such that γ to 1 iff ϕ holds, if ϕ is a sentence,
(ii) the function which maps every α such that γ to ϕ otherwise.

$$\lambda\alpha : \gamma.\phi = \begin{cases} 1 \Leftrightarrow \phi & \text{if } \phi \text{ is a sentence} \\ f : \alpha \to \phi & \text{otherwise} \end{cases}$$

So the lexical entry (2) for an intransitive verb, can be rewritten more concisely as a λ-expression, in which case the denotation of *smiles* is characterized as the (characteristic function of the) set of all individuals who smile, according to the first case in definition 2.3.

$$\begin{aligned} [\![smiles]\!] &= f : D_e \to D_t \text{ such that } \forall x \in D_e, f(x) = 1 \text{ iff } x \text{ smiles} \\ &= \lambda x \in D_e \ . \ x \text{ smiles} \end{aligned} \tag{2}$$

The lexical entry for a transitive verb in (3) can be written in the more compact λ-notation too, but in that case the denotation of *loves* is a function (to be precise, a function from individuals to functions from individuals to truth-values), and not a set as in the previous example, as described in the second case of definition 2.3:

$$\begin{aligned} [\![loves]\!] &= f : D_e \to \{g : D_e \to \{0,1\}\} \text{ such that} \\ &\quad \forall (x,y) \in D_e \times D_e, f(x)(y) = 1 \text{ iff } y \text{ loves } x. \\ &= \lambda x \in D_e \ . \ [\lambda y \in D_e \ . \ y \text{ loves } x] \end{aligned} \tag{3}$$

Functional application, one of the main composition mechanisms of our computational semantics system, is encoded as the well-known operation of *β-reduction* of λ-terms (see [3] for further details):

$$(\lambda\alpha.\phi)\beta = \phi[\beta/\alpha]$$

where the second member of the previous equation stands for "the term ϕ where all free[2] occurrences of α have been replaced by β". What follows shows the result of applying a function to an argument by a stepwise β-reduction:

[2] We use the word 'free' in the logical sense, as a not-quantified variable.

$$[\lambda x \in D_e \,.\, \lambda y \in D_e \,.\, y \text{ loves } x](\text{Mary})(\text{John}) \overset{\beta}{\Longrightarrow} [\lambda y \in D_e \,.\, y \text{ loves Mary }](\text{John})$$
$$\overset{\beta}{\Longrightarrow} 1 \text{ iff John loves Mary}$$

In this formal setting, Fregean "saturational" notion of meaning that we shortly mentioned in section 2.2.2 takes a precise computational meaning. Atomic semantic types e and t correspond to Frege's *saturated* denotations; besides those basic types there are various sorts of functions which correspond to Frege's *unsaturated* denotations. An intransitive verb has semantic type $e \rightarrow t$ and so it has an unsaturated place which must be filled by a individual of type e to yield a full sentence of type t. Analogously, a transitive verb is given semantic type $e \rightarrow e \rightarrow t$ and so it has two un-saturated slots that must be filled to yield a complete sentence. The typing machinery that our semantic model associates to expressions is not only a useful tool to better understand the semantic nature of linguistic objects we are dealing with, but also plays an important role in the computational process of interpretation, as it will be made more clear in the following section.

2.2.4 Composition Rules

The set of semantic entries of a lexicon is one of the two basic components of the interpretation function $[\![\ ]\!]$, the other being a mechanism to systematically combine those basic meanings associated to words into more complex expressions, that is, a set of *composition rules*. We assume that the input to function $[\![\ ]\!]$ is a phrase-structure like those presented in section 2.1.1. We adopt the two following conditions on interpretations for phrase-tree structures, globally referred to as the *locality principle* of interpretation:

Locality principle of interpretation: for any phrase-marker
- (i) every syntactic constituent X has an interpretation;
- (ii) in interpreting a constituent X, no other information can be used for computing $[\![\text{X}]\!]$ than that associated with X's daughters.

In [44] and [43], Richard Montague presents a system which defines semantic rules for specific types of subtrees. This approach was challenged by Klein and Sag in [35], in which they criticize the construction-specific method of classical Montague Grammar and propose (in a more genuinely Fregean spirit) typed functional application as the only semantic composition rule. More recent developments in the field of formal semantics (in particular see [26]) take a less radical approach and identify a limited number of semantic composition rules, in which functional application still plays a predominant role. We present some of the most common composition rules that we will make use of throughout the rest of the present work.

Terminal Nodes (TN rule): if α is a terminal node, $[\![\alpha]\!]$ is specified in the lexicon.

$$\begin{aligned}
[\![John]\!]^{e} &= \text{John} \\
[\![cat]\!]^{e \rightarrow t} &= \lambda x \in D_e \,.\, x \text{ is a cat} \\
[\![sleeps]\!]^{e \rightarrow t} &= \lambda x \in D_e \,.\, x \text{ sleeps} \\
[\![likes]\!]^{e \rightarrow e \rightarrow t} &= \lambda x \in D_e . \lambda y \in D_e . y \text{ likes } x
\end{aligned}$$

An important exception to this rule is represented by pronouns and traces, whose semantics is not provided by the lexicon but by the context of utterance. We'll devote section 2.2.5 to this important matter.

Non-branching Nodes (NN rule): if α is a non-branching node, and β is its daughter node, then $[\![\alpha]\!] = [\![\beta]\!]$. This amounts to state that labels in phrase-markers play no role in the semantic interpretation process.

$$\left[\!\!\left[\begin{array}{c} \text{VP} \\ | \\ \text{V} \\ | \\ \text{sleeps} \end{array} \right]\!\!\right] = \left[\!\!\left[\begin{array}{c} \text{V} \\ | \\ \text{sleeps} \end{array} \right]\!\!\right] = [\![sleeps]\!]$$

Functional Application (FA rule): if α is a branching node and $\{\beta, \gamma\}$ is the set of α's daughters, then

$$[\![\alpha]\!] = [\![\beta \cdot \gamma]\!] = \begin{cases} [\![\gamma]\!]([\![\beta]\!]) & \text{if } [\![\beta]\!] \text{ is in the domain of } [\![\gamma]\!]; \\ [\![\beta]\!]([\![\gamma]\!]) & \text{if } [\![\gamma]\!] \text{ is in the domain of } [\![\beta]\!]; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$\left[\!\!\left[\begin{array}{c} \text{S} \\ \text{NP} \quad \text{VP} \\ | \quad\quad | \\ \text{N} \quad\quad \text{V} \\ | \quad\quad | \\ \text{Ann} \quad \text{sleeps} \end{array} \right]\!\!\right] \overset{\text{(by FA)}}{=} \left[\!\!\left[\begin{array}{c} \text{VP} \\ | \\ \text{V} \\ | \\ \text{sleeps} \end{array} \right]\!\!\right] \left(\left[\!\!\left[\begin{array}{c} \text{NP} \\ | \\ \text{N} \\ | \\ \text{Ann} \end{array} \right]\!\!\right] \right)$$

$$\text{(by NN)} = \left[\!\!\left[\begin{array}{c} \text{VP} \\ | \\ \text{sleeps} \end{array} \right]\!\!\right] \left(\left[\!\!\left[\begin{array}{c} \text{NP} \\ | \\ \text{Ann} \end{array} \right]\!\!\right] \right)$$

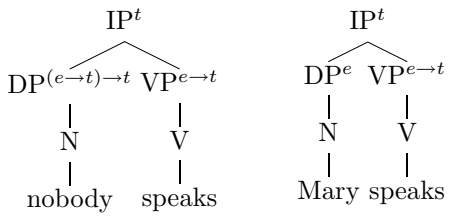
$$\text{(by NN)} = [\![sleeps]\!]([\![Ann]\!])$$

$$\text{(by TN)} = (\lambda x \in D_e . x \text{ sleeps})(\text{Ann})$$

$$\text{(by } \beta\text{-reduction)} = 1 \text{ iff Ann sleeps}$$

Note that FA rule *does not* specify the linear order of β and γ. Otherwise stated, the semantic interpretation component can ignore certain features that syntactic phrase structure trees are usually assumed to have. Just like syntactic category labels (see TN rule), linear order is irrelevant for semantic interpretation. According to such a *type-driven* interpretation procedure, it's the semantic type of the sister constituents which determines the mode of semantic composition, rather than their syntactic category and/or their linear order. So it's the type information that we introduced in definition 2.1, and that we can easily compute in a simply typed λ-calculus based semantic formalism, which will guide the interpretation algorithm, instead of additional information about syntactic category of the components involved, or about the linear order.

The following example, which also illustrates the basic idea behind the treatment of quantifiers in our semantic theory, elucidates the role of type-driven approach in the semantic interpretation process:



$[\![speaks]\!] = \lambda x \in D_e$. such that x speaks
$[\![nobody]\!] = \lambda f \in D_{e\to t}$. there is no $x \in D_e$ such that $f(x) = 1$

In the second parse the functional category is the VP, in the first one that same VP is the argument of functional category DP. What justifies this different compositional behavior between the two sentences and drives the interpretation algorithm for the whole expression is the type associated to the VP. In the first parse it fits the argument slot for the type $(e \to t) \to t$ DP, while it acts as a functor for the DP in the second parse. We do not need to specify any side-condition to justify the different treatment, because the semantic types of the constituents drive the interpretation process.

2.2.5 Pronouns, Traces and Variables

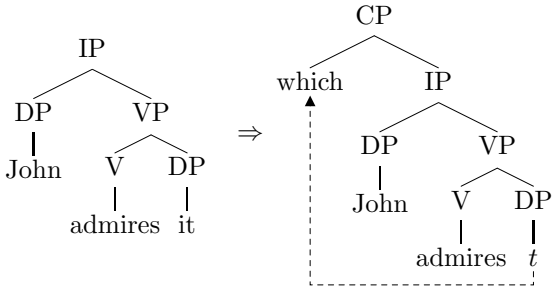
The focus of this thesis is the computational treatment of Binding Theory, which deals with sentences containing pronominal expressions like (4):

(4) John kissed her.

In order to interpret (4) we have to know what *her* denotes. Semantically, *her* behaves like a *free variable*, that is, something which derives its content from the context. Variables were introduced to "behave" like ordinary referring phrases, and therefore to denote an individual, but only *relative* to the choice of a particular assignment. This is the case of (4), where it's the context in which the sentence is uttered that provides the necessary information to recover the semantic content of pronoun *her*.

In any syntactic theory issued from the generative tradition *traces* play an important role. Putting details aside, we can say that almost any syntactic theory recognizes inside the sentence structure the presence of "movements", that is operations by which a word or a constituent is moved from one position in a structure to another to fulfill some kind of linguistic constraints. A trace is a phonetically empty category left behind (as a result of movement) in each position out of which a constituent moves. This movement does not come without consequences: a syntactic dependency gets established between the trace and the moved constituent, with important consequences for the semantic interpretation. An example of such movements is the parse proposed in example (5) for the expression *which John admires*, in which the final phonetic realization of relative clause results from the movement of *which* from the object position in the fragment *John admires it* leaving phonetically empty trace t in the original position.

(5)



It is clear from example (5) that in order for the semantics of the sentence to work we must assume that traces, just like proper nouns and pronouns, bear semantic type e. Semantically, as it will be made clear in the rest of this section, traces behave like *bound* variables.

To handle pronominal and movement constructions, we associate an *index* to every pronoun and every trace, and we have to augment the interpretation function $[\![\ ]\!]$ with an *assignment* function g, that is, a function from the set of natural numbers to individuals (the set of assignments is thus the set $D_e^{\mathbb{N}}$). When an expression contains a trace or a pronoun, the interpretation function $[\![\ ]\!]$ can be actually computed only with respect to an assignment g, and so it should be written as $[\![\ ]\!]^g$. We add to the current set of interpretation principles the following rule for interpreting traces and pronouns:

Traces and Pronouns (TP rule): if α is a pronoun or a trace, g is a variable assignment, and $i \in dom(g)$, then $[\![\alpha_i]\!]^g = g(i)$.

(6) a. $[\![he_2]\!]^{\left[\begin{smallmatrix}1 \to Sue\\ 2 \to Joe\end{smallmatrix}\right]} = \begin{bmatrix}1 \to Sue\\ 2 \to Joe\end{bmatrix}(2) = Joe$

b. $[\![t_1]\!]^{\left[\begin{smallmatrix}1 \to Sue\\ 2 \to Joe\end{smallmatrix}\right]} = \begin{bmatrix}1 \to Sue\\ 2 \to Joe\end{bmatrix}(1) = Sue$

The TP rule deals with pronouns and traces in exactly the same way: both are type e syntactic elements which get their denotation from the context in which the sentence is uttered. To lighten up the notation, we adopt the following convention:

$$[\![\alpha]\!]^a = \begin{cases} a & \text{if } \alpha \text{ is a trace or a pronoun} \\ [\![\alpha]\!] & \text{otherwise} \end{cases}$$

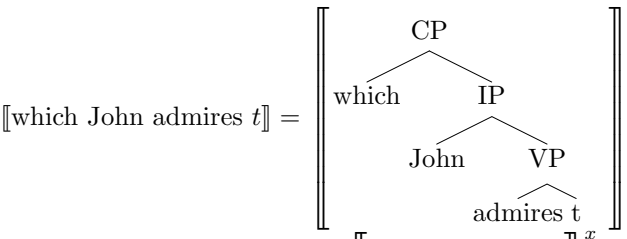
So for example $[\![t]\!]^{\text{MARY}} = \text{MARY}$ but $[\![sleeps]\!]^{\text{MARY}} = [\![sleeps]\!]$.

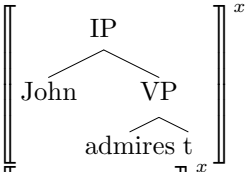
We have already said that the movement which led to the formation of the constituent *which John admires* in (5) establishes a syntactic dependency between the moved constituent and its trace. The next interpretation rule makes this dependency explicit at the semantic level:

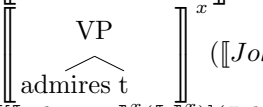
Predicate Abstraction (PA rule): if α is a branching node whose daughters are a relative pronoun and β then $[\![\alpha]\!] = \lambda x \in D_e.[\![\beta]\!]^x$

This rule has the effect of turning a trace (if it occurs in β) into a variable which is bound by the λ operator at the semantic level. Example (7) presents an occurrence of this rule at work:

(7)

$[\![\text{which John admires } t]\!] =$ 

(by PA) $= \lambda x.$ 

(by FA) $= \lambda x.$ 

$([\![John]\!]^x)$

(by TP + FA) $= \lambda x.[[[\![admires]\!]^x([\![t]\!]^x)](\text{John})]$

(by TP) $= \lambda x.[[[\![admires]\!](x)](\text{John})]$

(by lexical entry) $= \lambda x.[[[\lambda y.\lambda z.z \text{ admires } y](x)](\text{John})]$

(by β-reduction) $= \lambda x.[[\lambda z.z \text{ admires } x](\text{John})]$

(by β-reduction) $= \lambda x.$ John admires x

Predicate Abstraction rule provides an example of *syncategorematic* treatment for the moved relative pronoun. Syncategorematic items do not bear a semantic denotation of their own, but their presence affects the calculation of the semantic value for the next higher constituent. This kind of approach goes against the principles of type-driven interpretation and more theoretically adequate treatments of relative pronouns can be given.

2.2.6 Determiner Phrases

Binding Theory deals with the distribution and the semantic content of linguistic elements known as Determiner Phrases. This class includes, among others, proper names (*John*, *Mary*, etc.), pronouns (*he*, *she*, *them*, *his*, *her*, etc.), and so-called definite descriptions (*the woman who married Bill*, *the king of France*, etc.). For our purposes it is convenient to associate semantic type e to every DP, be it a pronoun, a proper name, or a definite description. Intuitively, this amounts to say that the denotation of a DP is an individual, or entity. Although an intuitive and unharmful operative choice for our enquiry, this is not precise from the point of view of formal semantics.

Quantificational expressions like *every linguist*, *three students* belong to the syntactic class of Determiner Phrases too, but we have seen that they are given semantic type $(e \to t) \to t$. That is, they do not denote individuals, but higher order predicates, or properties of properties. As such, they seem to be intrinsically different from type e entities. This superficial analysis, that assigns different semantic types to elements belonging the same grammatical category of DPs, runs into several problems. To name only one, in a sentence like *John and three students*

came to the party it is not possible to assign a semantic type to an expression like *John and every student* which coordinates seemingly different objects like an entity (*John*) and a set of properties (*every student*). One of the major achievements of Montague's work [43] on the formalization of semantics for fragments of English was the insight that proper names and definite descriptions can be seen as generalized quantifiers too, therefore providing a uniform semantic characterization of all DPs as denoting type $(e \rightarrow t) \rightarrow t$ objects. According to this view, the denotation of proper name *John* is not a particular individual in the domain of discourse, but the set of properties which are true for a particular individual. This uniform treatment of DPs perfectly accounts for coordinated structures like *John and every student* keeping the traditional semantic type $t \rightarrow t \rightarrow t$ for conjunction *and*. Formally:

$$\begin{aligned}
[\![John]\!] &= \lambda P.[P(\textsc{john})] \\
[\![John\ walks]\!] &= [\lambda P.[P(\textsc{john})]](\lambda x.\textsc{walk}(x)) = \textsc{walk}(\textsc{john}) \\
[\![every\ student]\!] &= \lambda P.[\forall x.\textsc{student}(x) \rightarrow P(x)] \\
[\![John\ and\ every\ student]\!] &= \lambda P.[P(\textsc{john}) \wedge [\forall x.\textsc{student}(x) \rightarrow P(x)]]
\end{aligned}$$

The same can be said for definite descriptions like the infamous *the king of France*, for which we can adopt the traditional Russellian analysis:

$$[\![the\ king]\!] = \lambda P.[\exists x.[\textsc{king}(x) \wedge \forall y.[\textsc{king}(y) \rightarrow y = x] \wedge P(x)]]$$

This homogenous treatment of DPs opens up other possibilities, namely a sound treatment of intensionality, which is one of the strengths of Montague's grammar.

However, for the purposes of our work, this sophisticated semantic analysis of DPs does not have any meaningful advantage over the naïve view of DPs as entities, not more than in everyday life the perception of the full range of radio frequencies is more useful than the limited sensibility to visible light spectrum. Therefore, we'll assume that definite descriptions denote type e entities, and we'll stick to this assumption throughout the rest of the present work. The semantics we'll officially assume for the definite article *the* in particular will be as follows:

$$[\![the]\!]^{(e \rightarrow t) \rightarrow e} = \lambda f : \exists!x[f(x)].[\iota y[f(y) = 1]]$$

where $\exists!x[\phi]$ abbreviates $\exists x[\phi(x) \wedge \forall y[\phi(y) \rightarrow x = y]]$ (i.e. there exists one and only one x such that $\phi(x)$), and $\iota x[\phi]$ selects the unique x such that $\phi(x)$.

2.2.7 Quantifier Raising

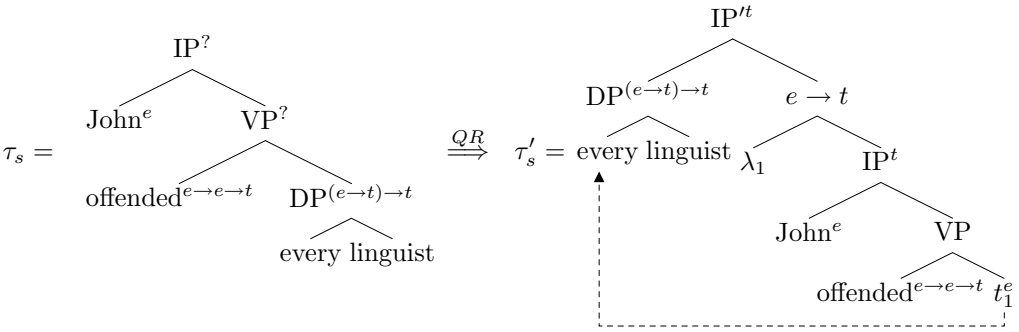
Quantifier Raising (QR) will play a central role in the integrated computational treatment of Binding Theory that we are going to present in chapter 4. We shortly recall here the basic ideas that will be of interest to our computational purposes.

Quantifier Raising has been developed explicitly within frameworks that assume that a semantic type is associated with each semantic expression. It was originally introduced as a syntactic covert (i.e. without phonetic realization) movement which restructures a phrase-marker in order to amend a semantic type mismatch between its constituents. When, in a given phrase-marker τ_s, one of the DP

constituents α is Quantifier Raised (or QR'd), α is moved to a higher position (or landing-site) in the parse tree, it is replaced by a trace t_i and it adjoins α to a dominant node. In order to implement the syntactic dependency existing between the trace and the moved constituent, the sister constituent of α is a lambda operator coindexed with the trace t_i. More formally:

$$\text{QR} : [_{IP} \ \dots \alpha \dots \ ] \Rightarrow [_{IP'} \ \alpha \lambda_i [_{IP} \ \dots t_i \dots \ ] \ ]$$

Example below shows the graphical notation for QR movement (inspired by [26]) that we will use throughout this chapter:



In τ_s the quantificational DP *every linguist* is semantically translated as a generalized quantifier, and as such it has type $(e \to t) \to t$: it's a function from predicates (functions from entities to truth values) to truth values. The transitive verb *offended* is a two-place predicate, and thus it has type $e \to (e \to t)$: it's a function from entities to functions from entities to truth values. Therefore, in τ_s no functional application is possible between *offended* and *every linguist*. The mismatch is amended in τ'_s: the DP constituent is raised above the IP node, leaving the trace t_1, which is supposed to bear semantic type e. The node labeled with λ_1 in τ'_s marks the introduction of an operator of lambda abstraction which binds the variable associated to t_1.

In the computational treatment of Binding Theory that we will present in chapter 4, we are going to make use of Quantifier Raising in a new way, in addition to the traditional one as a syntactic tool to amend semantic type mismatch. Although QR was first introduced to (covertly) move Quantificational Phrases (that is, type $(e \to t) \to t$ constituents), we license Quantifier Raising also for type e DPs, namely pronouns and proper names. In doing so, we adhere both to theoretical and computational convenience criterions. Reinhart in [50] provides convincing evidence that also type e DP can undergo QR, basically to solve the sloppy-strict identity puzzle for elliptic construals. Furthermore, it can be proved that truth conditions of a sentence in which a type e DP has been quantifier-raised are equivalent to those in which the DP is left *in situ*. As pointed out in [26], in this situation it is difficult to see what could forbid the possibility that DPs of type e undergo the same movements of DPs of type $(e \to t) \to t$.

Quantifier Raising is generally considered as the main operation of interest in transformational derivations from Surface Structure to Logical Form, and to apply *optionally* and *freely* to (particular types of) DPs. We instead adhere to the idea of Danny Fox (see [20]) of Quantifier Raising as a "last resort" tool which is

triggered on demand by specific computational pressure. Such driving forces may be the need to amend a semantic type mismatch or, as it the case case of the algorithms that we are going to present in chapter 4, to generate a bound-variable reading between two Determiner Phrases.

3

Binding Theory for Dummies

Go fuck yourself, Mr. Cheney!

A Katrina survivor,
quoted in The Washington Post, September 8th, 2005.

3.1 Introduction

Consider the following sentences:

(1)
a. Ann blames herself
b. John admires him
c. *John admires John
d. John thinks that Ann admires him
e. *Ann claims that John blamed herself
f. The woman that married John likes him
g. Every professor thinks that he is underpaid
h. No woman admits that she is a bad driver

There is no doubt that *herself* in (1-a) refers to Ann; just like there is general agreement that whoever *him* refers to in (1-b), this (male, singular) person cannot be John. On the other hand it could very well be the case that *him* in (1-d) refers to John, but that's not the only option, especially if the sentence is uttered in a context where John is not the only male individual, and the same holds for (1-f). Everybody agrees that (1-e) is plain wrong, and that *herself* cannot refer neither to *Ann* nor to *John*, although an English speaker feels that the reason why *herself* cannot refer to the former is not the same why it cannot refer to the latter. But who does *he* refer to in (1-g), and who does *she* to in (1-h)? And on the basis of which principle, syntactic or semantic in nature, we perceive (1-c) as wrong, or at least highly unlikely?

Binding Theory (BT for short) aims at describing the formal principles that govern such kind of judgements. That is, not only judgements on the well- or

ill-formedness of sentences which contain pronouns, but also on the relationship between their denotations. Whether such judgements are syntactic or semantic in nature is still a highly debated subject.

More technically, we can say that Binding Theory deals with the distribution and the referential properties of Determiner Phrases (or DPs) occurring in a sentence. We will see in the next sections how this characterization of BT is somewhat imprecise, and how the interpretation of its objectives and principles has evolved throughout last the 30 years of linguistic and semantic enquiry since its first formulation in the seminal works of Chomsky [12] and Lasnik [38], [39] at the beginning of the 80's.

3.2 Coreferential Binding Theory

Nearly all approaches to Binding Theory in English partition the set of DPs into three disjoint categories (we indicate between parentheses the traditional, although not universally accepted, terminology due to the enormously influential work of Chomsky [12] and the following generative tradition):

Reflexive pronouns and reciprocals (or anaphors): *himself*, *herself*, *itself*, *ourselves*, *themselves*, *each other*, *one another*, etc.;

Non-reflexive pronouns (or pronominals): *he*, *she*, *it*, *us*, *them*, *his*, *her*, *our*, *him*, *them* etc.;

Full-DPs (or r-expressions): *John*, *the king of France*, *the man who married Mary*, *a cat*, *every woman*, etc.

Anaphors and pronouns can be identified on the basis of morphosyntactic criteria, while we may negatively characterize full-DPs as every DP which is neither an anaphor nor a pronoun. There is an important semantic difference between the first two classes of DPs and the third one. A full-DP bears an independent semantic content which is either directly provided by the lexicon, or computed from the denotation of its constituents. Instead, semantic interpretation for pronouns and anaphors depends on the denotation of other entities that belong to the sentence in which they occur, or to the discourse context in which the sentence is uttered, with which they are said to be *coreferential*. In order to compute the correct interpretation for a sentence, a speaker must be able to recover their semantic content, and BT aims at identifying the general principles that govern this process commonly carried out very quickly by a human speaker.

3.2.1 Indexing, c-command, syntactic binding

Binding Theory was first formulated as a module of Government and Binding Theory by Chomsky in [12] (see also [7] for an updated account on BT). It relies on a syntactic device called *coindexing*, and on a structural relation between nodes of the phrase-marker of a sentence called *c-command*.

In the original formulation *indexing* is the practice of denoting the interpretation relations existing among the DPs of a sentence by means of indexes, that is integers attached to DPs. We adopt the following *convention on DP indexing*:

- every anaphor, pronoun and definite DP bears an index;
- nothing else bears an index.

Sameness of reference or denotation between two DPs is encoded by *coindexing*, that is by the two DPs carrying the same index. As a convention, we assume that two DPs corefer if and only if they are coindexed.

(2) a. John_1 thinks that Mary likes him_1.
b. John_2 thinks that Mary likes him_1.

In (2-a) *him* is coindexed with *John*, and therefore they are supposed to have the same denotation: on the basis of the convention just introduced, we can say that for all i, $[\![John_i]\!] = [\![him_i]\!]$). In (2-b) *John* and *him* are *contra-indexed*, which implies that their denotations are different: for all i, j, $i \neq j \Rightarrow [\![John_i]\!] \neq [\![him_j]\!]$.

If we consider indexes to be part of the syntax, certain configurations of indexes associated to the DPs occurring in a sentence give rise to syntactically ill-formed sentences. (Un)grammatical representations for these sentences will look like the following ones:

(3) a. [Zelda]$_2$ bores [herself]$_2$
b. *[Zelda]$_1$ bores [herself]$_2$
c. [She]$_4$ adores [Zelda's]$_7$ dog
d. *[She]$_4$ adores [Zelda's]$_4$ dog
e. [Zelda]$_2$ adores [her]$_2$ dog
f. [Zelda]$_2$ adores [her]$_7$ dog

Indexes allow us to express in a concise way the basic phenomena which Binding Theory accounts for. If we restrict our attention to the case of singular DPs, two DPs in a given sentence can show one of three logically possible coreference patterns:

(4) a. *Obligatory coreference*:
[Zelda]$_\text{x}$ bores [herself]$_\text{y}$ ($x = y \Rightarrow [\![Zelda_x]\!] = [\![herself_y]\!]$)
b. *Obligatory non-coreference*:
[She]$_\text{x}$ adores [Zelda's]$_\text{y}$ dog ($x \neq y \Rightarrow [\![she_x]\!] \neq [\![Zelda_y]\!]$)
c. *Optional coreference*:
[Zelda]$_\text{x}$ adores [her]$_\text{y}$ dog

The key insight of Binding Theory is that the (un)availability of coreference between two DPs depends on (but may not be limited to) two factors:

- the morphological shape of the DPs
- the structural relation between the DPs

The effect of morphological features on blocking coreference between two DPs is easily understood: in order for two DPs to have the same denotation, they must have compatible agreement features (namely, gender and number). However, as it is apparent from the contrast between (4-b) and (4-c), morphological features are necessary but not sufficient to account for different patterns of coreferentiality. This is why we focus our presentation on the *structural relations* that license or

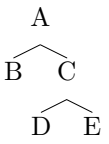
forbid coreference patterns between DPs, and most of our examples will involve pairs of DPs with compatible agreement features.

The structural relation that the classical formulation of Binding Theory identifies as the key feature in licensing or forbidding coreference between DPs is *c-command* between nodes of a phrase-marker. Given the phrase-marker τ for a sentence, we say that a node n_1 *c-commands* another node n_2 in τ when n_1 does not dominate n_2 and the first node dominating n_1 also dominates n_2:

Definition 3.1. *Node A* c-commands *node B in a tree if and only if*

(i) neither dominates the other;
(ii) every node that dominates A also dominates B.

More intuitively, we can say n_1 c-commands n_2 if either n_2 is a sister of n_1, or it's the descendant of a sister of n_1.

(5)


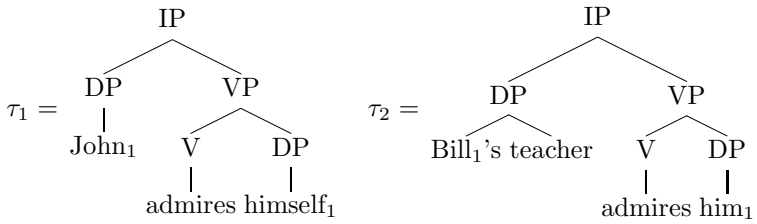
In (5) B c-commands C, D and E; C c-commands B; D c-commands E; E c-commands D; no other c-command relation exists.

Coindexing and c-command are the two notions on which the relation of *syntactic binding* is grounded. Two DPs are said to be (syntactically) *bound* when one c-commands the other and they are coindexed:

Definition 3.2. DP_1 is a binder for *(or* binds*)* DP_2 *if and only if*

(i) DP' *and* DP'' *are coindexed;*
(ii) DP' *c-commands* DP''*.*

Although the previous definition is universally referred to as "syntactic binding", it is actually a notion that lies at the frontier between syntax and semantics. On the syntactic side, it requires a purely structural (and thus syntactic) relation to hold between DP nodes of a phrase-marker; on the semantic side, it requires sameness of denotations between them, encoded by coindexing, which is a formal device which "imports" semantics into syntax.



In τ_1 *John* c-commands and is coindexed with *himself*, therefore it binds it. In τ_2 *Bill* and *him* are coindexed but they're not in a mutual relation of c-command, therefore no binding occurs between them.

3.2.2 Principles A, B, C

Traditional Binding Theory characterizes the configurations in which two DPs occurring in a sentence can, must or must not be coreferential on the basis of the notion of binding. Therefore, the core of BT consists of three principles that describe as many configurations in which a DP can, must or must not be bound within a sentence.

Principle A. A reflexive pronoun must be bound within its local domain.

(6) a. John$_1$ hates himself$_1$.
b. John$_1$ thinks that Bill$_2$ hates himself$_2$.
c. *John$_1$ thinks that Bill$_2$ hates himself$_1$.

Principle B: A non-reflexive pronoun must not be bound within its local domain.

(7) a. *John$_1$ hates him$_1$.
b. John$_1$ hates him$_2$.
c. *John$_1$ thinks that Bill$_2$ hates him$_2$.
d. John$_1$ thinks that Bill$_2$ hates him$_1$.

Principle C: A referential expression must not be bound.

(8) a. *John$_1$ hates John$_1$.
b. John$_1$ hates John$_2$.
c. He$_1$ hates John$_2$.
d. *He$_1$ hates John$_1$.
e. *He$_1$ thinks that Bill$_2$ hates John$_1$.

3.2.3 On local domain

Principles A, B and C are three conditions that constrain the coreferential configurations for three disjoint classes of DPs. The general form of a principle is:

(9) A DP of **class** $\left\{ \begin{array}{c} \text{must} \\ \text{must not} \end{array} \right\}$ be coindexed with a **c-commanding** DP within its **local domain**.

The class and the c-command status of a DP can be unambiguously established. The notion of local domain of a DP is much more difficult to characterize and deserves some further explanation.

As a first rough approximation, we define the local domain of a DP as the *smallest clause* that it belongs to. For example, the local domain for the anaphor or the pronoun occurring in (6-a), (7-a), (7-b), (8-a), (8-b), (8-c) and (8-d) is the whole sentence; for (6-b), (6-c), (7-c), (7-d) and (8-e) it is the clause introduced by complementizer *that*.

However, there are several examples in English that challenge this tentative characterization of local domain of a DP. We are going to detail only one particularly meaningful class of them, namely the case of Exceptional Case Marking constructions, that gives a fairly precise perception of the complexity of this subject.

Exceptional Case Marking (ECM) constructions are introduced by specific ECM-verbs (like *believe*, *want*) that take an infinitive complement (which is basically an embedded clause) whose subject is case-marked as objective by the ECM-verb.

(10) a. John believes him to be wrong
b. Mary wants her to leave

Since we defined the local domain of a DP as the smallest clause that contains it, in this situation we're on a deadlock, since it seems like *him* in (10-a) and *her* in (10-b) belong to two different clauses which only partially overlap. The following schema from Büring [7] summarizes the contrast:

(11) a. $\overbrace{[S^m \text{SUBJ}^m \dots V [S^e \text{SUBJ}^e}^{\text{binding domain for SUBJ}^e} [V\ OBJ]]$
b. $[S^m \text{SUBJ}^m \dots V \underbrace{[S^e \text{SUBJ}^e [V\ OBJ]]}_{\text{binding domain for OBJ}}$

The phenomenon is correctly captured by Binding Theory principles is we assume as local domain the *governing category* for a DP:

Definition 3.3. *The Governing Category for a DP is the smallest clausal category which dominates:*

- *DP*
- *DP case assigner*

Such a definition of local domain solves the puzzle of (11-a). The embedded object OBJ receives its case from the embedded V, hence its governing category is the embedded clause. Accordingly, the embedded object must be reflexive if coreferent with the ECM-subject, but not if coreferent with the matrix subject. The ECM-subject receives its case from the matrix verb, which means that the embedded clause is not its Governing Category: it contains the ECM-subject but it does not contain its case assigner. The smallest clausal category that fulfills the two requirements in the definition of GC is the matrix clause, and coreference of the ECM-subject with the matrix subject requires the former to be reflexive.

3.2.4 Summary on the coreferential approach

The classical perspective on Binding Theory presumes a tight correspondence between linguistic expressions and real world entities. Principles A, B and C define conditions that a sentence must fulfill in order to be well-formed. Through indexes, which encode semantic information in the syntax, such well-formedness syntactic principles happen to constrain the denotation of the DPs occurring in a sentence, thus ultimately interacting with the denotational level of representation. As we will see in the following section, this view has been severely challenged both on theoretical and empirical grounds by alternative interpretations that have been given to the principles of BT.

3.3 Reinhart's Bound-Variable Approach (1983)

Tanya Reinhart in [50] and [51] challenges the idea that Binding Theory rules *coreference* relations among the DPs occurring in a sentence. She basically claims that only one type of coreference relation is syntactically represented and directly constrained by the principles of grammar, that is one that stems from the well-known relation of *variable binding* in the sense of formal logic. Other coreference conditions are not even represented at any syntactic level, and as such they can be neither licensed nor forbidden on the basis of structural constraints. They are subject instead, Reinhart argues, to an extragrammatical principle that states that the coreferential interpretation is unavailable whenever the same meaning can be conveyed by means of variable binding.

3.3.1 Semantic Binding vs. Coreference

As Daniel Büring puts it in [7], *coreference* is only one of two semantic concepts that fall under the pre-theoretic concepts of *binding*, the other being *variable binding*. Thus the formal device of coindexing that we introduced in section 3.2.1 sometimes encodes sameness of reference and sometimes variable binding. It is thus necessary to clearly keep these two notions separated. We illustrate this basic difference with the help of the following examples:

(12) a. Coreference: $\left\{ \begin{matrix} He_1 \\ John_1 \end{matrix} \right\}$ said that he_1 was okay.

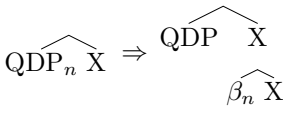
b. Variable binding: $No\ woman_2$ doubts that she_2 is okay.

The DPs in (12-a) display a coreferential behavior, in the sense that they both refer to the same entity in the real world (i.e. they have the same denotation) encoded by coindexing. This is not *necessarily* true: with a different indexing the second occurrence of pronoun *he* may very well refer to some other singular male individual previously introduced in the discourse domain. In particular, if *John* and *he* have the same denotation, *he* can be replaced by *John* without affecting the overall semantics of the sentence. This is actually the case: sentences $John_1$ *said that* he_1 *was ok* and $John_1$ *said that* $John_1$ *was ok* are semantically equivalent (modulo a Principle C violation in the second sentence).

This is not the case for (12-b). By replacing she_2 with $no\ woman_2$ we get a whole different meaning for the sentence. The fact is that *no woman does not refer* at all to a specific individual, and thus the non-reflexive pronoun *she* a fortiori *cannot corefer* with it. It turns out that there is a whole class of DPs (such as Quantifier DPs and wh-phrases) which are not referential and thus cannot support coreference. A more thorough analysis shows that, within the c-commanding domain of a Quantified DP with index n, pronouns bearing the index n are no longer referring pronouns, but bound pronouns. Their behavior is entirely similar to the behavior of a bound variable in a logical language: their value is no longer determined by contextual assignment, but by the argument slot filled by the DP. We adopt the following interpretation rule on the semantic treatment of Quantified DPs (issued from the systematization operated by Daniel Büring in [7] and Heim and Kratzer in [26]):

Index Transfer Rule:

For any Quantified Determiner Phrase QDP with index n, adjoin β_n to QDP's sister constituent:



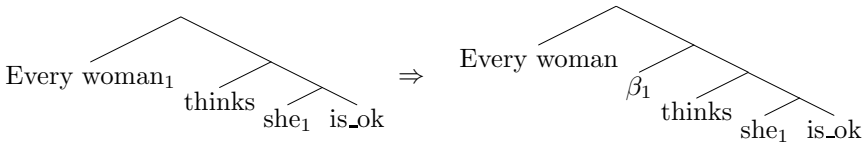
The adjoined β_n is called the *binder prefix* and the DP that minimally c-commands it is called the *semantic binder*. The following rule implements the semantic interpretation for binder prefixes:

Binder Prefix Interpretation Rule:

For any natural number n, $\left[\!\left[\widehat{\beta_n \ \mathrm{Y}} \right]\!\right]^g = \lambda x.[[\![Y]\!]^{g[n \to x]}(x)]$

The Interpretation Rule formalizes the intuition that β_n's sister constituent Y must not be interpreted relative to the original assignment g, but to a modified assignment $g[n \to x]$ where every occurrence of n has been replaced with x. The index n of QDP does not encode coreference: it is a device to specify that any pronoun indexed with n occurring in QDP's c-command domain (its sister subtree) must be interpreted as a bound variable. Since x is also the individual argument to $[\![Y]\!]$, the overall effect is the binding of any pronoun bearing the index n in Y by the argument slot of Y.

(13) $[_{\mathrm{QDP}}$ Every woman $]_i$ thinks that she$_i$ is ok



If we apply Binder Prefix Interpretation Rule to the second parse tree we get:

$$\begin{aligned}
\lambda x.[[\![\textit{ thinks that } x_1 \textit{ is ok }]\!]^{g[x_1 \to x]}(x)] &= \lambda x.[[\lambda y.\textsc{think}(y, \textsc{ok}(x_1))]^{g[x_1 \to x]}(x)] \\
&= \lambda x.[[\lambda y.\textsc{think}(y, \textsc{ok}(x))](x)] \\
&= \lambda x.[\textsc{think}(x, \textsc{ok}(x))] \\
[\![\textit{every woman}]\!] &= \lambda Q[\forall y.\textsc{woman}(y) \to Q(y)] \\
[\![\textit{every woman thinks that she is ok}]\!] &= \forall y.\textsc{woman}(y) \to \textsc{think}(y, \textsc{ok}(y))
\end{aligned}$$

which is the correct first order predicate logic representation of the meaning of the sentence.

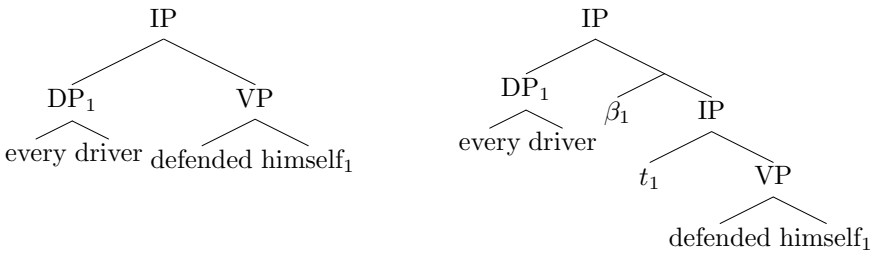
Note that if a pronoun is in the c-command domain of a QDP but it is not coindexed with it, it will still be interpreted as a free pronoun. The Interpretation Rule just introduced crucially relies on replacing every occurrence of the coindexed pronoun with a bound variable: when the pronoun is not coindexed (e.g. in *every woman$_1$ thinks that she$_2$ is ok*) it remains a free pronoun whose denotation is independent from the QDP's and which receives its value from the context.

On the basis of the formal apparatus just introduced to deal with QDPs we introduce the notion of *semantic binder*:

Definition 3.4. *A binder prefix β semantically binds a DP α if and only if:*

1. *α and β are coindexed*
2. *β c-commands α*
3. *there is no binder prefix β' which is c-commanded by β and meets (1) and (2)*

It is worth stressing the difference between syntactic and semantic binding of a pronoun:



Pronoun *himself* is syntactically bound in both parse trees, but it's semantically bound only in the second structure. This entails that in the first parse tree *himself* is interpreted as a free variable, whose semantic content is provided by a variable assignment supplied by the utterance context; in the second parse tree the pronoun is interpreted as a bound variable in the sense of predicate logic.

3.3.2 Principles A, B (and C) revisited

The distinction between coreference and semantic binding was introduced with the help of QDPs, which show more clearly the features and implications of variable binding. However, there's no reason to prevent a type e DP from acting as a semantic binder too. Consider the following example:

(14) a. she$_1$ knows her$_1$ rights
b. she [β_1 [knows her$_1$ rights]]
c. [$\lambda x.x$ knows x's rights](x_1)

By applying Index Transfer to (14-a) read as (14-b) we get the bound-variable reading (14-c) as its semantic interpretation. Reinhart claims in [52] that all pronouns can be interpreted as bound variables, regardless of whether the antecedent is a quantified DP or not, subject to the bound anaphora condition.

Reinhart's approach to Binding Theory is based on the notion of *semantic binding* in opposition to *syntactic binding* issued from the coreferential approach:

Semantic Binding: a DP α semantically binds a DP β if and only if α is the sister of a binder prefix γ for β.

In (14-a) personal pronoun *she* semantically binds *her* if we assume the semantic reading in (14-b). It is apparent from this definition that if α binds β this entails that α c-commands β. Therefore c-command is relevant for the syntactic conditions under which λ-predicates can be formed compositionally, but has no independent role in defining binding as in the coreferential approach.

The conclusion drawn by Reinhart is thus the following: all Binding Principles, and thus Binding Theory as a whole, regard semantic binding only. This is a direct consequence of Reinhart's hypothesis that only semantic binding is represented in the syntax. Coreference is a phenomenon taken care of at a different level of description, namely pragmatics or discourse theory. On the basis of this approach Binding Theory principles must be revised in the following way:

Principle A:
A reflexive pronoun must be semantically bound within its local domain.
Principle B:
A non-reflexive pronoun must be semantically free in its local domain.

Some phenomena that were previously accounted for by Condition C effects in coreferential Binding Theory are now subsumed by an interface principle that will be introduced in the next section.

(15) a. John likes himself
b. $[\lambda x.\textsc{like}(x, x)](\textsc{john})$

(16) a. John likes him
b. $\textsc{like}(\textsc{john}, x)$

(17) a. John thinks that Bill hates him
b. $[\lambda x.\textsc{think}(x, \textsc{hate}(\textsc{bill}, x))](\textsc{john})$
c. $\textsc{think}(\textsc{john}, \textsc{hate}(\textsc{bill}, x))$

In (15-a) principle A forces semantic binding between *John* and *himself*: the result is the logical form (15-b), which entails a bound-variable reading between the denotations of the two DPs. In (16-a) principle B blocks semantic binding between *John* and *him*. This forbids $[\lambda x.\textsc{like}(x, x)](\textsc{john})$ as a possible semantic interpretation for the sentence. If *him* cannot but be interpreted as a free pronoun, the correct logical form is (16-b). In (17-a) principle B states that no semantic binding can occur between *Bill* and *him*, which belong to the same local domain, while nothing forbids semantic binding between *John* and *him*, which belong to different domains. Therefore both logical forms (17-b) and (17-c) are suitable semantic interpretations for (17-a).

Note that in (16-b) and (17-c) nothing prevents x from being mapped into JOHN: semantic binding (and therefore BT as a whole, Reinhart argues) is basically about relations between *variables*, while nothing is said about the relations between their *values*. If this does not seem disturbing for (17-c), it has the somewhat odd effect of licensing $[\textsc{like}(\textsc{john}, x)]^{x \to \textsc{john}} = \textsc{like}(\textsc{john}, \textsc{john})$ as a possible interpretation for (16-b). This is exactly Reinhart's point: there are contexts in which such kind of assignment is perfectly legal. It is the case of so-called "Oscar sentences", that is conversational contexts like (18), in which it is perfectly possible that *she* and *her* have the same denotation:

(18) A: Is this speaker Zelda?
B: How can you doubt it? *She* praises *her* to the sky. No competing candidate would do that.

By providing an example of context in which such kind of assignment is perfectly acceptable, Reinhart makes a point in favor of the thesis that there is no true grammatical compelling reason to forbid coreference between a pronoun and a DP which locally c-commands it, but only to forbid a bound-variable reading between them. This is precisely the kind of phenomena that Binding Theory actually governs, while it has nothing to say about the semantic content of DPs.

3.3.3 Reinhart's Rule I

If we adopt Reinhart's approach to the principles of Binding Theory, nothing allows us to distinguish between (19-b) and (19-c) below as a semantic interpretation of (19-a):

(19) a. John thinks he is sick
b. $[\lambda x.\textsc{think}(x, \textsc{sick}(x))](\textsc{john})$
c. $[\textsc{think}(\textsc{john}, \textsc{sick}(x))]^{x \to \textsc{john}}$

Logical form (19-b) corresponds to the bound-variable reading, while (19-c) to the coreferential one. They are truth-conditionally indistinguishable and as far as we know they are perfectly equivalent. According to Reinhart, this is not true, but she must introduce an extra-grammatical principle on the basis of which establishing the difference:

Rule I (also known as the Coreference Rule):
A DP α cannot corefer with a DP β if an indistinguishable interpretation can be generated by (indexing and moving β and) replacing α with a variable bound by the trace of β.

Simply stated, if for a given sentence we can generate two semantic interpretations which are truth-conditionally equivalent, but such that one involves semantic binding and another coreferential mechanism, the bound-variable reading must be preferred. By introducing this rule, we are forced to choose (19-b) as the correct semantic reading for (19-a). This is motivated by Reinhart by means of some kind of "economy" principle. Semantic binding is an internal, purely grammatical device by which a human speaker can achieve coreferential effects. It is thus the preferred option when a speaker is in a situation like (19-a) as a semantic device to convey the intended meaning. The empirical evidence that forces Reinhart to introduce Rule I is more evident if we consider the following cases:

(20) a. John likes him
b. *$[\lambda x.\textsc{like}(x, x)](\textsc{john})$
c. $\textsc{like}(\textsc{john}, x)$

(21) a. John likes himself
b. $[\lambda x.\textsc{like}(x, x)](\textsc{john})$

We already know that Principle B forbids (20-b) as a possible interpretation for (20-a), while (20-c) is perfectly acceptable. Variable x being a free variable, there is no constraint whatsoever on the value it may take: suppose that for a given assignment $g : x \mapsto \textsc{john}$. Coreference rule implicitly states that this is not an

acceptable semantic reading for (20-a), thus forbidding such kind of assignment. Indeed, logical forms $[\textsc{like}(\textsc{john}, x)]^{x \rightarrow \textsc{john}}$ and $[\lambda x.\textsc{like}(x, x)](\textsc{john})$ are perfectly equivalent from a truth-conditional perspective. This means that Rule I kicks in and selects reading (20-b) which involves semantic binding. But this binding configuration corresponds to sentence (21-a). Otherwise stated, if a speaker wanted to convey the meaning $\textsc{like}(\textsc{john}, \textsc{john})$, he would choose the surface structure (21-a) instead of (20-a).

The coreference rule is used by Reinhart to eliminate an important part of BT, that is Principle C, which is actually made superfluous. Consider the sentences:

(22) a. *he$_1$ likes John$_1$
b. *he$_1$ likes John$_1$'s mother

They must be considered as ill-formed because we could replace *John* by a variable semantically bound by *he*, in accordance with Rule I:

(23) a. John$_1$ likes himself$_1$
b. John$_1$ likes his$_1$ mother

Sentences (23-a) and (23-b) have the same truth-conditions as the corresponding (22-a) and (22-b), but they avoid coreference in favor of semantic binding by means of the reflexive predicate and thus, by Rule I, they are chosen by a human speaker and the alternative ones must be ruled out. The obviative phenomena formerly explained by an additional syntactic principle are thus subsumed by the more general "pragmatic" principle conveyed by Rule I.

3.3.4 Summary of Reinhart's Approach

Reinhart's bound-variable interpretation of the principles of Binding Theory advocates a view in which BT has nothing to say about the relationship between linguistic expressions and real world entities like in the classical coreferential approach. Instead, Binding Theory governs a purely formal and "grammatical" property like semantic binding between DPs. Otherwise stated only semantic binding is represented in the syntax, and thus ruled by Binding Theory; coreference is a whole different phenomenon which is out of the reach of syntax and that must be dealt with at a pragmatic/discourse theory level of analysis. Although semantic binding can have coreference effects as a by-product, they must be kept conceptually distinct from what Reinhart calls "accidental coreference" effects, which are basically due to the context of utterance of the sentence and that therefore lie out of reach of grammar principles. In the language of logic, we may say that BT defines structural constraints on the variables occurring in a formula, but not on the values that they may take by means of assignment functions.

3.4 Reinhart and Reuland's "Reflexivity" Approach (1993)

Tanya Reinhart and Eric Reuland propose in [54] and [53] another perspective on the traditional tenets of Binding Theory. The basic evidence their approach accounts for is that local anaphors allow logophoric use, that means (roughly stated) that in particular contexts they are used as free variables. Their analysis acknowledges that a distinction is needed between logophoric processes and structural binding relations. In order to do that they distinguish three domains of anaphors: local, medium-distance, and logophoric. Binding Theory, from the point of view of Reinhart and Reuland, only deals with the first two.

This approach advocates a return to a somewhat traditional analysis in terms of *reflexivization*, as a property of predicates. The basic idea is that a reflexive pronoun is used in the object position when the verb expresses a reflexive relation. The focus of the theory is therefore on the nature of the relation expressed by the verb. The semantic shift initiated in [50] is carried to the extreme consequences: the structural notion of c-command definitely loses the central role it bears in the classical Chomskian formulation of Binding Theory, and the focus of the theory is not on coreference or semantic binding phenomena anymore, but on the reflexive nature of predicates. Principles A and B do not define anymore constraints on the distribution or denotation of pronouns (although such constraints result as by-products of the theory), but are reformulated as conditions on reflexive predicates and the morphosyntactic features that mark them as such.

3.4.1 Problems with the standard binding conditions

Traditional Principles A and B rely on a distinction perceived as fundamental between anaphors and pronouns. The basic assumption is that there is a perfect complementarity in distributions between these two classes of DPs, which is highlighted by their traditional formulation:

Principle A: an anaphor is bound in its local domain.
Principle B: a pronoun is free in its local domain.

Otherwise stated, a pronoun is not allowed in an environment allowing an anaphor. This is confirmed in examples like:

(24) a. Max criticized himself / *him.
b. Max speaks with himself / *him.

But this is contradicted by examples like:

(25) a. Max saw a gun next to him/himself.
b. Lucie saw a picture of her/herself.

The basic observation to be made here is that examples in (24) and (25) differ in their *argument structure*. In (24) the anaphor and its antecedent are coarguments, while in (25) they are not. Reinhart and Reuland generalize this observation by stating that *pronouns are disallowed only when the pronoun and its antecedents are coarguments*. The environments where a pronoun must be free are much more restricted than the environments where an anaphor can be bound. The use of

anaphors in contexts like (25) is an instance of their *logophoric* use: the neat separation of such kind of phenomena, which are ruled at a discourse theory level, from those which are governed at the grammatical level, and which only are the subject of Binding Theory, is one of the major achievements of this new approach.

3.4.2 The Basic Database

Anaphoric expressions are divided into pronouns or anaphors. What sets the former apart from the latter is a property called Referential Independence (R). Pronouns enjoy +R property (which they share with full-DPs) and thus can be used demonstratively, referring to some entity in the real world. Anaphors are referentially defective (-R property) and therefore can't: binding is the procedure which assigns the content necessary for their referential interpretation.

Anaphors are further divided into long-distance anaphors (SE-anaphors: Dutch *zich*, Norwegian *seg*, Italian *sé*, etc.) and local anaphors (SELF-anaphors: English *himself*, Dutch *zichself*, Italian *se stesso* etc.). Although structurally identical to pronouns, SE-anaphors differ from them in that they lack full specification of so-called θ-features like number, gender, person: this is taken as the property which accounts for their defective, and thus anaphoric, nature which they share with SELF-anaphors. However, the two kinds of anaphors have a different status as far as it concerns their Reflexivizing function: SELF-anaphors have the property of making reflexive (i.e. imposing the identity on two arguments of) the predicate they apply to, which SE-anaphors do not have. Both types of anaphors admit a logophoric use.

The map of the properties is summarized by Reinhart and Reuland in the following schema:

	SELF	SE	Pronouns
Reflexivizing function	+	−	−
Referential independence	−	−	+

The fundamental claim of Reinhart and Reuland is that reflexivizing and referential functions of DPs are governed by distinct modules of linguistic knowledge, which together capture the full distribution of local anaphora. What matters in binding conditions is actually related to the Reflexivizing property of expressions, whereas all aspects of their distribution which are sensitive to the R property fall under Chain Theory. Binding Theory focusses on the grammatical function of anaphors, and thus or their reflexivizing status.

3.4.3 Reflexivity

A predicate denotes a reflexive relation if and only if two of its arguments are identical, like in:

(26) $\lambda x.P(\ldots, x, \ldots, x, \ldots)$

Predicates of the form (26) are called *reflexive predicates*. One of the tenets of Reflexivity approach is the basic observation that only reflexive pronouns make a transitive predicate reflexive when they occur in argument position (this is the meaning of the property +R associated only to SELF anaphors in the table in section 3.4.2).

(27) a. Lucie$_i$ adores herself$_i$
b. Lucie's$_i$ joke about herself$_i$

In (27-a) *herself* occurs as one of the arguments of predicate *adores*, thus making it reflexive. In (27-b) we assume that *herself* is an argument of a 2-places predicate *joke*, the other place being saturated by the author of the joke, therefore making it a reflexive predicate too.

Note that the definition of reflexive predicate is given in terms of identity of *variables* (i.e. identity of positions in the grid structure, or signature of the predicate) bound by the same lambda operator, not of *values* of the variables. We have already met this important distinction in section 3.3. Identity of variables is encoded in the signature of the predicate and establishes a necessary identity between the arguments that occur in those positions. Identity of values may be an accidental phenomenon, just like in the sentence *she praises her to the sky* of example (18) the two pronouns may refer to the same entity in the real world, but there's no necessary identity condition for the two arguments encoded in the structure of the predicate *praises*. The reflexivizing function of reflexive pronouns operates at the level of the signature, or the predicate's grid, by identifying *two positions* (i.e. variables), and not only their values.

SELF-anaphors form reflexive predicates only when they occur in co-argument positions with other DPs. The following sentences are examples of situations in which SELF-anaphors *do not* form reflexive predicates:

(28) a. Lucie saw a picture of her$_i$/herself$_i$ in the paper
b. Max$_i$ saw a snake near him$_i$/himself$_i$

Neither in (28-a) nor in (28-b) the reflexive occurs in argument position for the predicate *saw*: in (28-a) the anaphor is embedded in the unary predicate *picture*, in (28-b) within the predicate *near*. We recall that the idea behind Reflexivity approach is that binding conditions only govern the *reflexive use* of anaphors. In (28-a) and (28-b) the anaphors fail to make the predicate reflexive and thus no Binding Theory principle operates on either of them: that's the reason why in such contexts anaphors and pronouns are basically interchangeable (modulo agreement features). This shift challenges the basic assumption of traditional Binding Theory according to which distribution of SELF-anaphors is fully governed by condition A, but it accounts for the large empirical evidence which proves that logophoric anaphora is quite common also with SELF-anaphors.

3.4.4 Reflexive Marking

Reinhart and Reuland's Reflexivity approach to Binding Theory is founded on the assumption that a universal property of natural language is that reflexivity of a predicate must be explicitly licensed. A predicate is reflexive, i.e. it denotes the relation in (26), only if it is *marked as reflexive* at a morphosyntactic level.

Predicates can be marked as reflexive intrinsically or extrinsically. Intrinsically reflexive-marked predicates have their grid specified as reflexive in the lexicon and they can be used only reflexively. Intrinsic reflexivity may or may not be morphologically marked as such. Reinhart and Reuland claim that for example clitics (Italian *si*, French *se*) must be considered as morphological markers of intrinsic reflexivity instead of as SELF-anaphors. Neither in English nor in languages with clitics are found pronouns which saturate an intrinsically reflexive predicate.

Extrinsically marked predicates are transitive predicates which get reflexive-marked whenever a SELF-anaphor occurs at one of their grid positions. The main difference with SE-anaphors is that the latter lack this reflexivizing function, as already stated in section 3.4.2. This notion is generalized by the following definition:

Reflexive-marking
A predicate P is reflexive-marked if and only if either P is intrinsically reflexive or one of P's arguments is a SELF-anaphor.

In the following sentences the occurrence of a SELF-anaphor in argument position marks the predicate as reflexive:

(29) a. Lucie$_i$ adores herself$_i$
b. Lucie's$_i$ joke about herself$_i$

By focussing on the grammatical function of anaphors, the domains of anaphors occurrence can be reduced to just two. The ancient notion of "local domain" now corresponds to the domain of reflexivity, where a SELF-anaphor mandatorily reflexivizes a predicate, and where both pronouns and SE-anaphors are excluded. This is the domain of restated Principles A and B.

3.4.5 Conditions A and B

The revised versions of principles A and B proposed by Reinhart and Reuland establish a bidirectional relation between reflexive marking and reflexivity of a predicate.

Condition B: A reflexive predicate is reflexive-marked.

(30) a. *John$_1$ likes him$_1$.
b. John$_1$ likes himself$_1$.
c. *John$_1$/*he$_1$ likes John$_1$.
d. John$_1$ thinks that Mary likes him$_1$.

Coindexing between the two DPs in co-argument positions (i.e. they're arguments of the same predicate) in (30-a), (30-b) and (30-c) implies that *criticize* is a reflexive predicate in any of the three sentences. But only in (30-b) the presence of a SELF-anaphor in argument position actually reflexive-marks it and thus makes it compliant with the Condition B just introduced. The same holds for (30-c): neither *criticize* is an intrinsically reflexive predicate, nor any of its arguments is marked as a SELF-anaphor. So Condition B is not met and the sentence (with the given indexing) rejected. In (30-d) no predicate is reflexive because the coindexed DPs are arguments of different predicates: therefore Condition B does not apply and the sentence with the given indexing is accepted.

It is important to highlight that this new perspective on Condition B makes no use of configurational relations like binding, c-command, or argument hierarchy. It is strictly a condition on reflexive predicates, regardless of their internal structure.

Pushing further the idea that Binding Theory governs predicate reflexivity instead of semantic binding or coreference between DPs, Reinhart and Reuland introduce a new condition which establishes a relationship between reflexivity and reflexive-marking in the inverse direction:

Condition A: A reflexive-marked predicate is reflexive.

Condition A (a) states that a predicate which is reflexively marked must be interpreted as reflexive and (b) it does not say anything about anaphors (SELF-marked linguistic elements) which do not occur in an argument position and thus that do not function as reflexive markers. Condition A is therefore twofold: on one hand it governs the interpretation process of reflexively marked predicates, on the other it "says nothing" about anaphors that do not occupy a co-argument position, which therefore do not mark as reflexive a predicate and are made available to be used deictically or *logophorically*. This is what enables Condition A to successfully account for contrasts like the following:

(31) a. Max boasted that the queen invited Lucie and himself for a drink
b. *Max boasted that the queen invited himself for a drink

Classical Binding Theory struggled against such kind of evidence, in which an anaphor appears to occur free in (31-a), while is blocked in (31-b), thus violating the assumption that the use of SELF-anaphors was completely ruled by principle A. According to the Reflexivity approach, what discriminated the use of the anaphor in (31-a) from the use in (31-b) is that in the latter the SELF-anaphor is the argument of the predicate *invited*. Therefore it reflexive-marks the predicate, which, according to Condition A, must be reflexive. But this entails a reading in which *the queen* should be identified with *himself*, which is impossible due to agreement features mismatch. By contrast in (31-a) the anaphor is embedded in the argument of the predicate: the argument of *invite* is *Lucie and himself* which *is not* a reflexive argument. So no reflexive-marking occurs for the predicate *invite*, Condition A does not apply and the anaphor can occur free (i.e. logophorically).

3.4.6 Syntactic and Semantic Predicates

The basic version of Conditions A and B in the Reflexivity approach is challenged by contrasts like the following:

(32) a. The queen$_1$ invited both Max and herself$_1$ to our party.
b. *The queen$_1$ invited both Max and her$_1$ to our party.

The predicate in (32-a) is not reflexive-marked (the SELF-anaphor is embedded into the argument *Max and herself*), thus Condition A does not apply and the SELF-anaphor *herself* is free to refer logophorically to *the queen*. In (32-b) the sentence with *the queen* and *her* coindexed is not acceptable. There is no reflexive marked predicate, so Condition A does not apply. As for Condition B (a reflexive predicate must be reflexive-marked), in order to block this sentence we must find a predicate which is reflexive and not reflexive-marked. This is not apparent at first sight, but Reinhart and Reuland solve this puzzle by interpreting the sentence in the following way:

(33) $[\lambda x.(\textsc{invite}(x, \textsc{max}) \wedge \textsc{invite}(x, x))](\textsc{the queen})$

In (32-a) one of the arguments of the new semantic predicate INVITE is appropriately realized in the syntax as a SELF-anaphor; hence, the translation yielding a reflexive predicate is allowed. But in (32-b) no argument is a SELF-anaphor in the syntax, so the reflexive translation is disallowed and the derivation is filtered out. Therefore, to capture cases like (32-b), Reinhart and Reuland are forced to stipulate a distinction between *syntactic* and *semantic* predicates. As we have just seen, Condition B must be allowed to operate on semantic predicates, i.e. at the stage of translating syntactic predicates into semantic ones: Condition B thus describes a condition on semantic reflexivization. The same does not hold for condition A:

(34) a. Max$_1$ said that the queen invited both Lucie and himself$_1$ for tea.
b. The queen invited both Max and myself for tea.

If, as in the case of Condition B, Condition A was a condition on reflexive markedness of semantic predicates, in both (34-a) and (34-b) the anaphors would mark as reflexive the instance of the semantic predicate INVITE of which they occur as arguments, thus incorrectly blocking them. It follows that Condition A is a condition on the *syntactic* marking of predicates, and thus operates at a syntactic level. Reinhart and Reuland are thus forced to introduce a somewhat artificial distinction between syntactic and semantic predicates, and to adapt the formulation of Conditions A and B to them:

Syntactic predicate: the syntactic predicate formed by a head P is (i) P, (ii) all its syntactic arguments, and (iii) an external argument of P (the subject).
Semantic predicate: the semantic predicate formed by P is (i) P and (ii) all its arguments at the relevant semantic level.

Conditions A and B are modified to take into account this fundamental distinction:

Condition A: a reflexive-marked syntactic predicate is reflexive
Condition B: a reflexive semantic predicate is reflexive-marked

3.4.7 Chain Condition

In chapter 2 we have seen that if we assume a syntactic vision in which constituents *move* from one position to another (e.g. in the case of Quantifier Raising), we also make the assumption that they leave in the position out of which they move a (phonetically empty) trace. In the recent generative tradition in syntax a moved constituent and its trace are said to form *a chain*. More explicitly, a moved constituent and its trace are two different links of a *movement chain*. The moved constituent is said to be the *head* of the associated movement chain, and the trace the *foot* of the chain. The general convention used in the literature is to mark the binding relation between a trace and its antecedent by attaching identical indexes to them. The definition given by Chomsky in [13] goes as follows:

Generalized chain definition:

$C = (\alpha_1, \ldots, \alpha_n)$ is a chain if and only if C is the maximal sequence such that

1. there is an index i such that for all j, $1 \leq j \leq n$, α_j carries that index;
2. for all j, $1 \leq j < n$, α_j governs α_{j+1}.

Simply stated, a chain is a sequence of constituents such that each link of the chain c-commands the following one. Reinhart and Reuland introduce a condition on chains to account for some hierarchical effects that cannot be explained by Reflexivity conditions alone. In particular, since the particular view of Binding Theory developed in Reflexivity acknowledges no role to c-command, there's no way to account for ill-formedness of a sentence like **himself likes John*. In order to account for such cases, the following condition is introduced:

General condition on A-chains

A maximal A-chain $(\alpha_1, \ldots, \alpha_n)$ contains exactly one link α_i that is both +R and case-marked.

We know from section 3.4.2 that an item which enjoys the +R property is a referential element. With respect to the minimal database we're considering here, +R constituents are only full-DPs or non-reflexive pronouns used deictically. What matters to our purpose of a computational treatment of Binding Theory is the fact that Chain Condition accounts for some hierarchical effects previously taken care of by c-command requirements. Consider the following examples:

(35) a. He$_1$ criticized himself$_1$
b. *Himself$_1$ criticized him$_1$

Both (35-a) and (35-b) equally fulfil both Condition A and Condition B. So there's no way to account for the ill-formedness of (35-b) by means of a violation of one of the conditions. By definition the A-chain domain of a given DP is a subset of its c-command domain. It follows that the Chain Condition imposes hierarchical requirements on the relations of referential dependency within the A-chain. The referentially independent (+R) element of the chain must be its head; that is, it must c-command the referentially dependent (-R) element. In (35-a) this requirement is fulfilled, while in (35-b) it is not because the chain is headed by *himself* (-R element) and tailed by *him* (+R element).

3.4.8 Summary on Reflexivity approach

Conditions A and B taken together establish a correspondence between (semantic) reflexivity and (morpho-)syntactic reflexive-markedness. The central notions that found this approach to Binding Theory are those of argument and (marked) predicate. Anaphors are interesting only in their reflexive-marking features, but only when they occupy an argument position. Whenever they occur in a non-argument position, their behavior is predicted not to differ significantly from the one of another pronoun.

Classical binding theory was focussed on the strategic distinction between reflexive pronouns, non-reflexive pronouns and full-DPs. This new perspective stresses the role of argument positions, in which case an anaphor has the power to license reflexivity for a predicate, but also states that Binding Theory has nothing to say about anaphors occurring in non-argument positions, which are then free to assume logophoric or deictic uses, as correctly predicted by this approach. What the traditional view of BT struggled against due to empirical evidence of anaphors not occurring in traditional bound-variable contexts is one of the strengths of this approach.

3.5 On Reconstruction

3.5.1 Movement

Movement is the operation by which a linguistic constituent, or a set of features, is moved from one position (*base*) in a phrase-structure structure to another (*landing site*). Quantifier Raising of a Quantifier Phrase in object position in (36-a) and (36-b), and Wh-movement in (36-c) and (36-d) are examples of movement phenomena.

(36) a. John offended $[_{\text{QP}}$ every linguist$]_1$
b. $[_{\text{QP}}$ every linguist $]_1$ John offended t_1
c. You can speak $[_{\text{DP}}$ which languages$]_1$?
d. $[_{\text{DP}}$ Which languages$]_1$ can you speak t_1?

As shown in the examples above, a constituent which moves higher up in the syntactic structure leaves behind in the position out of which it moves an empty category, that is *a trace* of the previous constituent, which has exactly its same grammatical features but no phonetic content. The moved constituent is said to be the *antecedent* of its trace, and the antecedent of an empty trace is said to *bind* the trace. Using the relevant terminology introduced in section 3.4.7, a moved constituent and its trace together form a (movement) chain, and the moved constituent and its empty trace are said to be two links of the relevant movement chain. Just like in Binding Theory, the general convention used in the literature on trace theory is to mark the binding relation between a trace and its antecedent by attaching identical indexes to them.

We are not going to delve deeper into the details of trace theory: we'll only say that there is both theoretical and empirical evidence to support postulating the

existence of such phonetically empty categories. What is more interesting to our computational treatment of Binding Theory is that from a theory-internal point of view, trace theory enables us to explain otherwise puzzling properties of movement operations. The moved constituent always moves from a lower to a higher position in any given structure, and never from a higher to a lower one. Trace theory, together with the c-command condition on binding (see section 3.3.1), provides a natural explanation for the fact that movement is always upwards. If a moved constituent has to bind its trace, and if a bound constituent has to be c-commanded by its antecedent, it follows that a moved constituent must always move into a position where it c-commands (and thus it occurs higher up in the structure than) its trace: hence, the movement must always be in the upwards direction.

Two classes of movements are identified. Passivization and raising are different manifestations of a single argument-movement operation which has the effect of moving a constituent from one argument position into another. Such kind of movements globally go under the label of *A-movements*, because they involve movement to an argument position. On a complementary fashion, operations which move maximal projections into a non-subject position are instances of $\overline{A}$*-movements*, where the bar is given the typical linguistic interpretation of negation, meaning that they involve movements to non-argument positions.

3.5.2 Movement and Reconstruction

The first idea that led to the introduction of the notion of overt movement was that under particular circumstances a linguistic item is interpreted at a different position than where it is overtly (i.e. phonetically) realized. This idea has been challenged by evidence that proved, since the 1960s, that although some interpretative properties of a moved constituents (like those having to do with predicate-argument relations) are determined at the base position, other properties can be determined at the landing site (in particular scope and variable binding, see [10] for details). Under the pressure of such evidence the view of grammar architecture evolved towards a perspective in which overt movement affects meaning as well as sound. Quantifier Raising, as a movement operation invisible to phonology, was proposed as an account for scopal ambiguities within this framework. Structures involving movement were seen as feeding both the articulatory and the conceptual system (both LF and PF in the generative terminology). In the articulatory system the base position is not taken into account, while in the conceptual system both base and landing site are taken into account, and the interpretative properties appeared to be distributed between them.

A further step in the evolution of the notion of movement must be made to account for the fact that it seems that the effects of movement on scope and variable binding *are not obligatorily present*. It is now widely acknowledged that in some cases the semantic effects of movement are "undone" in a process known as *scope reconstruction*.

In order to account for facts related to scope reconstruction phenomena, two distinct approaches have been proposed. The first assumes that scope reconstruction is the outcome of semantic procedures. Within this approach, interpretative principles can deal with movement in at least two ways: one in which scope is

determined in the base position, the other in which scope is determined at the landing site. The second approach assumes that scope reconstruction is already determined in the syntax, or, otherwise stated, that the structures of LF, that serve as input to the semantic interpretation, determine whether or not there is scope reconstruction.

Fox argues in [19] that scope reconstruction is syntactic. This approach, in addition to having been proved linguistically sound, is also practically useful for a real world natural language understanding system. Most probably, in any context in which semantic computations are needed, most of available information will be of a syntactic rather than semantic nature. In particular, this means that scope reconstruction should be represented structurally, i.e. that there is *syntactic* reconstruction and that Binding Theory applies at the level of Logical Form.

3.5.3 Semantic versus Syntactic Accounts of Scope Reconstruction

It is well known that scope can be significantly affected by overt movement. The following sentences (taken from [19]) provide an example of such interaction:

(37) a. John seems to a teacher [t to be likely to solve every one of these problems]
b. [Every one of these problems] seems to a teacher [t to be likely t to be solved t by John].

In (37-b) the embedded quantificational object is overtly displaced, and the result of this displacement allows it to receive wide scope relative to another scope-bearing element (the existential quantifier *a teacher*). The overall effect is that two readings are available for sentence (37-b): one in which there is a single teacher, and another in which for every problem a different teacher thinks that it can be solved. This second scope relation would have been impossible without overt movement, as demonstrated in (37-a): in this case only one reading is available, that is the one in which the existential quantifier has scope over the universal one (there is a single teacher for every problem).

More interestingly to our computational purposes, the same point can be made with respect to variable binding, which the following examples prove to be deeply affected by movement phenomena:

(38) a. *The teacher is expected by his_1 mother [t to encourage every boy_1]
b. Every boy_1 is expected by his_1 mother [t to be encouraged by the teacher].

In (38-a) the universal quantifier cannot bind a variable that is outside of its scope, while (38-b) overt movement gives the quantifier wider scope and allows it to bind the variable.

Experimental data prove that overt movement does not *necessarily* affect scope. In the following constructions scope may be construed in the base position or in any of the intermediate landing sites.

(39) a. $[\text{At least one soldier}]_1$ seems to Napoleon [t_1 to be likely to die in every battle].

b. [At least one soldier]$_1$ seems to himself$_1$ [t_1 to be likely to die in every battle].
c. [At least one soldier]$_1$ seems to his$_1$ commanders [t_1 to be likely to die in every battle].

(40) a. One soldier is expected by Napoleon [t to die in every battle].
b. One soldier$_1$ is expected by his commander$_1$ [t_1 to die in every battle].

In the (a) sentences the universal quantifier in the embedded clause can take scope over the matrix subject, because readings in which soldiers vary with the battles appear to be compatible with those sentences. On the other hand another reading is available, the one in which the matrix subject takes scope over the existential quantifier, thus giving rise to the implausible interpretation which asserts the existence of a single soldier who is expected to die in every battle.

Such kind of ambiguity cannot be accounted for by the availability of long-distance Quantifier Raising. The universal quantifier *can* move by QR over the existential quantifier, and we could think that it's the optionality of this movement which is the cause of ambiguity, just like it happens in sentences (37-a) and (37-b). But the very same argument could be used to account for ambiguity in the (39-b) and (39-c) sentences too: however, those sentences are not ambiguous. Their meaning is restricted to the implausible interpretation that results from assigning wide scope to the existential quantifier, and thus that there is a single soldier who dies in every battle.

While the hypothesis that QR is the source of ambiguity for (a) sentences leaves this restriction for (b) and (c) sentences unexplained, the assumption that scope reconstruction is the real phenomenon involved accounts for it straightforwardly. Scope reconstruction would imply that the existentially quantified expression (*one soldier*, *at least one soldier*) can be interpreted back on t_1 positions. But in sentences (39-b) and (39-c), the existential quantifier must bind a variable in a position outside its scope as determined by scope reconstruction, hence scope reconstruction is not a viable option for them. This problem does not exist for the (a) sentences, and thus we must assume that the interpretation with the universal quantifier bearing scope over the existential of the (39-a) sentence stems from a combination of scope reconstruction and short-distance QR, which make both of the readings available. The matrix subject receives scope in the position of t and the universal quantifier receives scope above this position (via QR).

Syntactic Accounts of Scope Reconstruction

If we adopt a syntactic approach to scope reconstruction, an ambiguous sentence like (39-a) comes as already disambiguated at the stage of LF, that is, of the interpretation process. In (41-a) the disambiguation is implemented with the Quantifier Phrase (QP) being in its surface position and binding a variable in the trace position. Under the second disambiguation in (41-b), the QP is on the intermediate trace position. The syntactic disambiguation procedures yields the following two LF structures:

(41) a. [someone from New York]$_1$ is very likely [t_1 to win the lottery]

b. (it) is very likely [[someone from New York] to win the lottery]

This syntactic reconstruction process can be implemented in several different ways (like Quantifier Lowering from [42], and the copy theory of movement in [14]). Beyond implementation details, the important idea is that all syntactic accounts share the assumption that scope reconstruction involves an LF structure in which the Quantifier Phrase is literally in the reconstructed position.

Semantic Accounts of Scope Reconstruction

Semantic approaches assume that syntactic reconstruction is not necessary for scope reconstruction: they assume that there is a *purely semantic* mechanism that yields the two interpretations of sentences such as (39-a) from a structure with no syntactic reconstruction such as (41-a).

The semantic nature of such mechanisms is developed within frameworks that assume that a semantic type is associated with each syntactic expression. Within such frameworks, another assumption is that the sister of a moved constituent is interpreted as a function that can be expressed with lambda abstraction over a variable in the trace position (see our description of QR mechanism in section 2.2.7). Such a variable is assumed to range over individuals (and thus being interpreted as a variable of type e), or over generalized quantifiers (that is, as a variable of type $(e \to t) \to t$). The two options are presented in the following sentences, where x is supposed to range over individuals, and Q over generalized quantifiers:

(42) a. [Someone from New York] λx (is very likely [x to win the lottery])
b. [Someone from New York] λQ (is very likely [Q to win the lottery])

In (42-a) the variable x is of type e, and the sister of the moved QP is the result of a lambda abstraction over the variable of the trace which yields a function from individual to truth values (that is a predicate, semantic type $e \to t$), and thus QP (type $(e \to t) \to t$) takes its sister as its argument. The result is an interpretation in which the existential quantifier (*someone*) has scope over the modal verb (*to be likely to*). In (42-b) the sister of the quantifier is interpreted as a function from generalized quantifiers to truth values (type $((e \to t) \to t) \to t$). In this case the QP is the argument and its sister the functor, and the resulting interpretation is one in which the modal verb has scope over the existential quantifier.

3.5.4 Principle C, Syntactic and Semantic Accounts

Danny Fox in [19] grounds his argument on the syntactic nature of scope reconstruction on a careful use of Principle C from Binding Theory. In the hypothesis that Condition C is sensitive to LF structures, if we consider the following scheme, where linear precedence stands for the c-command relation:

(43) $[_{\text{QP}} \ldots \text{expression}_1 \ldots]_2 \ldots \text{pronoun}_1 \ldots \text{t}_2$

we can formulate the following prediction:

(44) *Scope reconstruction feeds Condition C*: scope reconstruction should be impossible in the structural configuration in (43).

Reconstruction would imply the Quantifier Phrase to be semantically interpreted in position t_2, but in that case the r-expression will be bound by a pronoun, and in the hypothesis that Condition C apply to LF structures, that would entail a violation of Principle C of Binding Theory.

In order for this prediction to follow under the *semantic account* of scope reconstruction, we'd be forced to assume that Condition C makes reference to the semantic type of traces and that the LF structure in (43) is ruled out if and only if the semantic type of the trace is $(e \to t) \to t$. On the other hand a *syntactic account* of scope reconstruction does not ask for such kind of stipulation. Condition C receives the definition based on constructions without movement. Under the assumption that an interpretative principle such as Binding Theory is sensitive to LF structures, (44) follows. Danny Fox in [19] provides substantial evidence that prediction (44) holds: this advocates both for the syntactic account and the assumption that Condition C applies at LF.

Further predictions can be drawn from statement (44). With respect to configuration (43), QP is obliged to take scope over all of the scope-bearing elements c-commanded by the pronoun. On the other hand, it is not obliged to take scope over the scope-bearing elements that c-command the pronoun.

(45) Predictions made by (44)

a. In (46) QP must take scope over the scope-bearing element SB^1.
b. In (47) QP need not take scope over the scope-bearing elementSB^2.

(46) [QP ... r-expression$_1$ ...]$_2$ ... pronoun$_1$ ... SB^1 ... t$_2$

(47) [QP ... r-expression$_1$ ...]$_2$ ... SB^2 ... pronoun$_1$ ... t$_2$

The latter claim depends on the assumption, largely uncontroversial, that there is a position for reconstruction between SB^2 and pronoun$_1$.

3.5.5 Reconstruction and Binding Theory

Reconstruction plays a central role in linguistic theory and displays important interactions with binding issues that need to be addressed. In this section we show that our integrated algorithmic approach interacts quite naturally with the current view on scope reconstruction issues.

The bottom line of [19] is that Binding Theory only applies at the LF level. This translate into an architectural principle of division of labor between the syntactic and the semantic level that we would like to keep in our system. Fox's conclusions underpin with linguistic arguments this computational goal. He presents evidence that the predictions of Condition C come out right only if we assume that this condition (and thus of Binding Theory as a whole) applies to the structures that are interpreted. In order to do that he needs to account for those cases in which there seems to be evidence that Condition C applies at S-structure, like in the following examples (taken from [19]):

(48) a. *You bought him$_1$ every picture that John$_1$ liked.
b. *He$_1$ bought you every picture that John$_1$ liked

(49) a. [[Which picture that John$_1$ likes] [did you buy him$_1$ t]]?

b. [[Which picture that John$_1$ likes] [did he$_1$ buy you *t*]] ?

(50) a. [[Every picture that John$_1$ liked] [I bought him$_1$ t]].
b. [[Every picture that John$_1$ liked] [he$_1$ bought you t]].

We have already seen that in order to account for the semantics of sentences with Quantifier Phrases in object position, we must postulate a covert movement, that is Quantifier Raising of the QP. So, under classical assumptions about the nature of covert QR, the LF structures of the sentences in (48) are those in (50). With respect to condition C, these structures are identical to the S-structures representations in (49). How does it come that different S-structures with equivalent LF forms allow different coindexings, those in (48) being wrong and those in (49) being correct? At first sight, if BT principles like condition C applied only at LF, there would be no obvious way of accounting for the contrast without accepting the hypothesis that BT applies at S-structures as well.

Chomsky provides a way of explaining the contrast without giving up to the assumption that Condition C applies only at LF forms and not at S-structures as well. He suggests that $\overline{A}$ movement always leaves a *copy* of the moved constituent, and that this copy under certain circumstances yields a Condition C effect, even if Condition C applies only at the output of the movement. So the true result of QR for sentences in (48) is quite different from (50). In particular, it still has a copy of the moved constituent at the position of the trace, and it is the r-expression within this copy that yield a violation of condition C that makes the sentence incorrect. So the true LF forms issued from QR for sentences in (48) are

(51) a. [[Every picture that John$_1$ liked] [I bought him$_1$ [every picture that John$_1$ liked]]].
b. [[Every picture that John$_1$ liked] [he$_1$ bought you [every picture that John liked]]].

For sentences in (49) Chomsky claims that $\overline{A}$ movement applies prior to the insertion of the relative clause that contains the r-expression. Therefore in (49) the copy of the moved constituent does not yield a Condition C effect. So in the light of Chomsky's copy theory just presented, a possible derivation for sentence *which picture that John$_1$ liked did you buy him t_1* is:

(52) a. did you buy him$_1$ [which picture]?
b. [[which picture] [did you buy him$_1$ [which picture]]]?
c. [[which picture [that John$_1$ liked]] [did you buy him$_1$ [which picture]]]?

The difference between overt and covert movement under this proposal is related not to their respective ordering relative to binding theory but to their respective ordering relative to lexical insertion. Covert movement is never followed by lexical insertion and therefore never appears to circumvent a Condition C violation.

There are cases of overt movement similar to the cases of covert movement in that they are unable to circumvent a condition C violation like in the following contrast:

(53) a. [Which argument [$_{\text{AP}}$ *that John$_1$ made*]] did he$_1$ believe t?
b. *[Which argument [$_{\text{CP}}$ *that John$_1$ is a genius*]] did he$_1$ believe t?

Chomsky accounts for this contrast on the basis of a distinction between the timing of adjunct insertion, like in (53-a), and the timing of complement insertion, like in (53-b). According to this distinction, complements, in contrast to adjuncts, must be inserted prior to movement. From this follows that complements, such as the italicized phrase in (53-b), in contrast to adjuncts, such as the relative clause in (53-a), cannot condition C via overt $\overline{A}$-movement. Basically, we adhere to Chomsky's proposal in [14] to account for the contrast without the assumption that condition C applies at S-structure.

3.6 Computational Perspectives on Binding Theory

Binding Theory analyzes from a purely theoretical standpoint a class of problems which has considerable applicative interest. The anaphora (or coreference) resolution problem (see section 1.2.1) is still one of the toughest and most strategic task that modern natural language understanding systems must tackle. Oddly enough, very little interaction has taken place between these two domains: the research on computational approaches to coreference resolution is mostly committed to knowledge-poor, statistical-heuristic approaches (which as such make very little or no use of linguistic insights), and the Binding Theory community is mostly forgetful of computational, not to mention applicative, issues.

In the present section we shortly review the main results stemmed from what has been proved to be a fruitful interaction between these two domains. On one hand the few "Binding Theory aware" algorithms for coreference resolution (sections 3.6.1, 3.6.2, 3.6.4) have proved to perform very well with respect to heuristic approaches in spite of their relative simplicity. On the other hand mathematical tools issued from theoretical research on computation such as combinatory and complexity analysis (sections 3.6.5, 3.6.6) have shed new light on the traditional issues of Binding Theory.

3.6.1 Hobbs' Algorithm (1978)

One of the first (and best) methods to tackle the anaphora resolution problem from a linguistically aware point of view is Hobbs' algorithm (see [27], [28]). Strictly speaking Hobbs' algorithm does not deal with Binding Theory; however, it represents one of the first accomplished efforts made in the field of Natural Language Processing to take seriously linguistics in designing effective algorithms to perform anaphora resolution for real-world tasks. Hobbs' algorithm is based on various syntactic constraints on pronominalisation which are used to search a parse tree. The search is done in an optimal order in such a way that the DP upon which it terminates is regarded as the probable antecedent of the pronoun at which the algorithm starts.

Hobbs' algorithm searches parse trees looking for possible antecedents of a pronoun. The overall architecture involves starting at the DP immediately dominating the pronoun and then searching the tree in a specified order which tries to maximizes chances of finding the correct antecedents, and looking for the first match in gender and number. The algorithm operates as follows:

1. begin at the DP immediately dominating the pronoun;
2. go up the tree to the first DP or IP encountered: call this node X, and the path to reach it, p;
3. traverse all branches below node X and to left of path p in a left-to-right, breadth-first fashion, proposing as the possible antecedent any DP node encountered which has a DP or IP between it and X;
4. if X is the highest IP node in sentence, search previous trees, in order of recency, left-to-right, breadth-first; when a DP node is encountered it is proposed as an antecedent. If X is not the highest IP node in the sentence, continue to step 5;
5. from node X, go up to first DP or IP node encountered, call this X, and the path to it p;
6. if X is a DP node, and p does not pass through the Nominal Node that X immediately dominates, propose X as the antecedent;
7. search below X, to left of p, left-to-right, breadth-first, proposing any DP encountered as the antecedent;
8. if X is an IP node, search below X to right of p, left-to-right, breadth-first, but not going through any DP or IP, proposing DP encountered;
9. Go to Step 4.

Intuitively, when a pronoun is encountered, the algorithm (step 2) moves up to the nearest S or DP node (according to the rough approximation of "local domain" that we anticipated in section 3.2.2) that dominates the pronoun and (step 3) searches to the left of the pronoun for all DP nodes that are dominated by an intervening DP node to propose as antecedents. The algorithm then (step 5) proceeds up to the next DP or S node and (step 6) searches to the left of the pronoun for any DP node to propose as antecedent. At this level, (step 7) a search is also carried out to the right for DP nodes to be proposed as antecedents. This will handle cases of backwards pronominalization. However, this portion of the search is bounded; it does not seek antecedents below any DP or S nodes encountered (step 8). The search for c-commanding antecedents and antecedents for backwards pronominalization (step 9) continues in this fashion until the top S is reached. At this point, (step 4) preceding utterances in the discourse are searched, going from most recent to least recent. Each tree is searched in a left-to-right, breadth-first manner for DPs to propose as antecedents.

In spite of its simplicity and the rough approximation of the traditional version of Binding Theory, Hobbs reports an accuracy of 88.3% on one hundred examples taken from three different texts.

3.6.2 Ingria and Stallard (1989)

In their 1989 paper [29], Ingria and Stallard present an algorithm for handling bound anaphora, disjoint reference, and pronominal reference. Their approach is inspired by the works of Lasnik [37] and the "indexing scheme" of Chomsky [11], from which they draw the idea that syntax is responsible for assigning *possible antecedents to anaphors* (Principle A) and *impossible antecedents to pronouns* (Principle B). The algorithm is augmented with Tables of Coreference as those introduced by Jackendoff in [30]. The algorithm was implemented as part of a

real-world language understanding system, the BBN Spoken Language System (see [6]).

The algorithm applies to a completed parse tree issued from a standard generative analysis and traverses it in a left-to right, depth-first manner. The local domain for the node being processed is defined as the most immediately dominating finite clause (S or IP, according to different denotations in generative tradition) node or DP node (when minimality has been induced by the presence of a possessive). With respect to local domain "external" and "internal" nodes are defined: internal nodes are dominated by the current minimal domain node; external nodes c-command the current minimal domain node. The result of the core syntactic phase of the algorithm is then passed to a semantic interpretation component: this includes a first "structural semantics" phase (corresponding to the formal semantics introduced in chapter 2) whose output is an expression in a higher-order intensional logic; and then a "lexical semantics" phase whose output is a (possibly empty) set of expressions of another higher-order intensional logic which implements a knowledge representation language.

Essentially, the algorithm passes each node all the nodes by which it is c-commanded. These nodes are subdivided into two sets, those that are internal to the current minimal domain and those that are external. As each node is processed, a subroutine is called that dispatches on the category of the node and performs any actions that are appropriate. It is this subroutine that implements the pronominal reference mechanism properly.

Ingria and Stallard algorithm differs from Hobbs' under several important respects. The latter handles both intra-sentential and extra-sentential pronominal reference relations, while the former is only intended to handle intra-sentential cases. Ingria and Stallard's algorithm is based on a single-pass, depth-first, exhaustive traversal of the parse tree, while Hobbs' first walks down the tree, then up, and then back down and is not guaranteed to be exhaustive.

3.6.3 Lappin & Leass (1994)

One of the most advanced efforts to take advantage of linguistic insights to devise an algorithm for coreference resolution is presented by Lappin and Leass in their 1994 paper [36]. Their Resolution of Anaphora Procedure (RAP) contains the following main components:

- an intrasentential syntactic filter for ruling out anaphoric dependence of a pronoun on a DP on syntactic grounds;
- a morphological filter for ruling out anaphoric dependence of a pronoun on an DP due to non-agreement of person, number, or gender features;
- a procedure for identifying semantically empty pronouns;
- an anaphor binding algorithm for identifying the possible antecedent binder of a lexical anaphor (reciprocal or reflexive pronoun) within the same sentence;
- a procedure for assigning values to several salience parameters (grammatical role, parallelism of grammatical roles, frequency of mention, proximity, and sentence recency) for a DP. This procedure employs a grammatical role hierarchy according to which the evaluation rules assign higher salience weights to (i) subject over non-subject DPs, (ii) direct objects over other complements, (iii)

arguments of a verb over adjuncts and objects of prepositional phrase (PP) adjuncts of the verb, and (iv) head nouns over complements of head nouns;
- a procedure for identifying anaphorically linked DPs as an equivalence class for which a global salience value is computed as the sum of the salience values of its elements;
- a decision procedure for selecting the preferred element of a list of antecedent candidates for a pronoun.

What is most interesting to our purposes are the syntactic filter and the anaphor binding component.

The filter consists of six conditions for DP-pronoun non-coreference within a sentence. The agreement features of an DP are its number, person, and gender features. A phrase P is said to be in the argument domain of a phrase N if and only if P and N are both arguments of the same head. P is said to be in the adjunct domain of N if and only if N is an argument of a head H, P is the object of a preposition PREP, and PREP is an adjunct of H. P is in the DP domain of N if and only if N is the determiner of a noun Q and (i) P is an argument of Q, or (ii) P is the object of a preposition PREP and PREP is an adjunct of Q. A phrase P is contained in a phrase Q if and only if (i) P is either an argument or an adjunct of Q, i.e., P is immediately contained in Q, or (ii) P is immediately contained in some phrase R, and R is contained in Q.

Once introduced the basic terminology of Lappin and Leass algorithm, we can state the six conditions of the syntactic filter. A pronoun P is non-coreferential with a (non-reflexive or non-reciprocal) noun phrase N if any of the following conditions hold:

1. P and N have incompatible agreement features;
2. P is in the argument domain of N;
3. P is in the adjunct domain of N;
4. P is an argument of a head H, N is not a pronoun, and N is contained in H;
5. P is in the DP domain of N;
6. P is a determiner of a noun Q, and N is contained in Q.

To implement the Anaphor Binding Algorithm the notion of *higher argument slot* is introduced by the following hierarchy of argument slots: subj > agent > obj > (iobj|pobj). Here subj is the surface subject slot, agent is the deep subject slot of a verb heading a passive VP, obj is the direct object slot, iobj is the indirect object slot, and pobj is the object of a PP complement of a verb, as in put DP on DP. A noun phrase N is a possible antecedent binder for a lexical anaphor (i.e., reciprocal or reflexive pronoun) A if and only if N and A do not have incompatible agreement features, and one of the following five conditions holds:

1. A is in the argument domain of N, and N fills a higher argument slot than A;
2. A is in the adjunct domain of N;
3. A is in the DP domain of N;
4. N is an argument of a verb V, there is a DP Q in the argument domain or the adjunct domain of N such that Q has no noun determiner, and (i) A is an argument of Q, or (ii) A is an argument of a preposition PREP and PREP is an adjunct of Q.

5. A is a determiner of a noun Q, and (i) Q is in the argument domain of N and N fills a higher argument slot than Q, or (ii) Q is in the adjunct domain of N.

Salience weighting is accomplished using salience factors. A given salience factor is associated with one or more discourse referents. These discourse referents are said to be in the factor's scope. A weight is associated with each factor, reflecting its relative contribution to the total salience of individual discourse referents. Initial weights are degraded as new sentences are processed in the text. All salience factors that have been assigned prior to the appearance of the new sentence sentence have their weights degraded by a factor of two. When the weight of a given salience factor reaches zero, the factor is removed.

In [36], Lappin and Leass claim a performance of 86% of correctly identified antecedents for the 360 pronouns occurring in a text and an average improvement of 4% with respect to Hobbs' algorithm.

3.6.4 Giorgi, Pianesi, Satta (1990)

Giorgi, Pianesi and Satta in [48] and [22] embark in an endeavor somewhat similar to ours, that is to provide a computational account of BT. However, they stick to the classical coreferential approach stemmed from Chomsky's Government and Binding Theory [12]. They provide two algorithms, each of which takes as input a node corresponding to a DP in a phrase-marker and analyzes some specific relations between the input node and each node in the parse tree which c-commands it up to a certain specific domain. What is remarkable in their approach is a fairly sophisticated notion of binding domain that they adopt. They define a binary predicate *domain* on pairs of nodes in the parse tree which returns TRUE (i) if the first node is the least constituent such that either all the $\theta-$roles pertaining a lexical head are realized, or all the *grammatical functions* pertaining to the same lexical head are realized; (ii) if there is a node which is the *lexical governor* of the second node whose father is not the same father of the first one.

They provide two algorithms to implement principle A and B which they prove to be polynomial. Whereas previous works were mainly concerned with the computation of possible referents for all the DPs occurring in a sentence (and thus actual indexings), they are faithful to the letter of BT principles, which only restrict the search space for indexes selections, without actually providing them.

3.6.5 Fong's Combinatorial Analysis (1990)

One of the first theoretical results issued from a genuinely computational perspective on Binding Theory was provided by the 1990 article of Sandiway Fong [18]. In it he performs a combinatorial analysis of the basic tenets of the principles-and-parameters framework of Chomsky [12]. In particular he addresses the issue of "free indexation", the idea that at an early stage of analysis, indexes are assigned freely (i.e. randomly) to all Determiner Phrases occurring in a sentence. Once indexes have been assigned, principles A, B or C kick in and rule out the interpretations that do not meet them. One might wonder whether this, although inefficient and inelegant, is nonetheless a viable computational strategy to generate all and only correct indexings for a sentence.

Fong proves that free indexation produces a number of indexings which is exponential in the number of independent DPs in a phrase structure. This result is achieved by a reduction to a well-known problem of combinatorial partitioning. The basic observation is that the problem of free indexation can be expressed as the problem of assigning $1, 2, \ldots, n$ distinct indexes to n Determiner Phrases in a sentence. The general problem of assigning m distinct indexes to n DPs is isomorphic to the problem of partitioning n elements into m non-empty disjoint sets: each partitioned subset represents a set of DPs with the same index.

We quote from [18] the following table which eloquently shows how the number of indexings grows with the numbers of independent DPs:

DPs	Indexings	DPs	Indexings
1	1	7	877
2	2	8	4140
3	5	9	21147
4	15	10	115975
5	52	11	678570
6	203	12	4123597

Any brute-force algorithm which computes *all* free indexation configurations to later sieve the results through a module which checks each indexing to be "Binding Theory compliant" clashes against this fundamental complexity issue: the number of checks that should be performed quickly grows beyond control.

By pushing his analysis one step further, Fong traces the guidelines of the approach we will be pursuing throughout chapter 4. He points out that although a reliable parsing mechanism must be able to produce every possible indexing, it does not necessarily follow that a parser must enumerate *every indexing* when parsing a *particular* sentence. He actually devised a procedure that enables a parser to shortcut such a blind enumeration procedure, and to produce fewer indexings via an inductive, bottom-up approach which filters out impossible readings.

Analogously, the purpose of our algorithmic procedure integrated into computational semantics is to avoid the exhaustive exploration of the search space by means of inductive computation of intermediate correct readings for smaller constituents of a sentence. This would correspond to the "factoring" mechanism that Fong points out at as a way to explore the search space of indexing in a controlled fashion. As he points out, early elimination of ill-formed indexings depends crucially on a parser's ability to interleave binding principles with structure building, which is precisely the basic structure of the algorithm presented in section 4.5.

3.6.6 Ristad's Complexity Analysis (1993)

An ambitious and controversial work on the complexity issues of the human language faculty, and more specifically of anaphora resolution problem, is provided by Ristad in [56]. His purpose is to characterize the properties of natural languages in terms of complexity theory, and he proves his claim analyzing the anaphora resolution problem as a case study. His conclusion is that the problem of understanding anaphora is NP-complete: it can be computed in nondeterministic polynomial time and is also NP-hard, that is, as hard as the "hardest" problems in the class of problems solvable in nondeterministic polynomial time (see, e.g., [46] for a more formal introduction to these concepts).

Ristad structures his arguments in five rounds of what he calls a "complexity game", which is a contest between a *maximizer*, who tries to make natural languages as complex as possible, and a *minimizer*, who seeks to reduce the complexity as much as possible. In the first round, he provides an argument to demonstrate the NP-hardness of any language whose anaphora are required to agree in features such as number, gender, and so on with their antecedents. In the second round, this argument is refuted by the minimizer, who claims that the standard theory of how agreement works is wrong and proposes a new theory, under which anaphoric agreement is now recognizable in deterministic polynomial time. In the third turn, a new set of data leads to the central argument in the book, according to which the anaphora problem is NP-hard after all. The facts crucial to this argument have to do with obviation, that is violations of principles B or C as defined in section 3.2. The maximizer goes on to the fourth round, in which he presents another body of data (dealing with anaphora in elliptical structures) on the basis of which he argues that the anaphora problem is PSPACE-hard (i.e., as hard as the "hardest" problems in the class of problems solvable in deterministic polynomial space, a class which contains the class of NP-complete problems). The minimizer replies, in the fifth and final round, by arguing against the theory of ellipsis presupposed in this argument and proposes another theory, under which the part of the anaphora problem involving ellipsis is no more difficult than the part dealing with obviation that was argued earlier to be merely NP-hard. The game thus ends with the result that the English anaphora problem is NP-hard, but within the class of NP-hard problems, that is to say, NP-complete.

Although not uncontroversial (his results have been severely questioned in [49]), Ristad's analysis represents a fascinating approach to the complexity issues of anaphora resolution and thus on the computational treatment of Binding Theory.

3.7 Binding Theory and DRT

Binding Theory does not tell the end of the story about pronouns. Pronouns are not only linguistic devices which refer to entities previously introduced within the same sentence: they also strategically work as semantic glue among sentences in a text which contains several of them. While the former case goes under the denomination of *intrasentential anaphora*, the latter is known as *intersentential anaphora*. Pronouns are among the most important linguistic devices to pass information from one sentence to another in a complex text. However, modelling semantics of

intersententially anaphoric pronouns raises a number of important problems with respect to the standard first order logic representations of sentences.

We have already seen in the previous sections of this chapter how the most straightforward way to establish links between anaphoric pronouns and their antecedents is to translate the pronouns as variables bound by their antecedents. This approach does not work when the link crosses a sentence boundary, as in the infamous example (54):

(54) A farmer$_1$ owns a donkey$_2$. He$_1$ beats it$_2$.

What we would like our logical representation language to allow is to interpret the first sentence of this discourse as soon as it is uttered, and then later on, while processing the second sentence, establish the links between the pronouns and their intended antecedents. If we think of translating the indefinites by means of existential quantifiers with scopes extending beyond the sentence level, and then allow the variables for the pronouns to be captured by these quantifiers, we only shift the problem. At some point the scope of a quantifier has to be closed, but further on another pronoun may occur that has to be linked to the same antecedent. The bound variable approach to anaphora also fails for cases where a pronoun in the consequent of a conditional sentence is linked to an indefinite noun phrase in the antecedent of the conditional, as in (55):

(55) If a farmer$_1$ owns a donkey$_2$, he$_1$ beats it$_2$.

A possible approach here would be to view (55) as a combination of the noun phrases *a farmer* and *a donkey* with a structure containing the appropriate gaps for antecedents and pronouns, like in (56). This is the approach of quantifying-in, taken in traditional Montague grammar.

(56) If PRO$_1$ man owns PRO$_2$, PRO$_1$ beats PRO$_2$.

This approach does not work here, however. Quantifying-in the indefinite noun phrases in (56), i.e. in a structure that has the conditional already in place, would assign the wrong scope to the indefinites with respect to the conditional operator.

(57) Every man$_1$ who owns a donkey$_2$ beats it$_2$.

In this case as well, quantifying-in does not allow one to generate the most likely reading where the subject of the sentence has wide scope over the embedded indefinite.

Discourse Representation Theory (DRT) aims at providing a brand new language to formalize the meaning of pronouns (but not only) in such contexts. DRT (originally formulated by Hans Kamp in 1981 and further developed in [33]) is a formal semantic model of the processing of text in context which has applications in discourse understanding. DRT stems from Montague's model-theoretic semantics and addresses in particular a number of difficulties of traditional approaches at the level of the discourse, such as tense and aspect, as well as anaphora resolution.

The basic idea of the DRT approach is that a natural language discourse (a sequence of sentences uttered by the same speaker) is interpreted in the context of a representation structure. The result of the processing of a piece of discourse

in the context of representation structure S is a new representation structure S_0; the new structure S_0 can be viewed as an updated version of S.

The interpretation of indefinite noun phrases involves the introduction of discourse referents or reference markers for the entities that a piece of discourse is about. Discourse referents are essentially free variables. Thus, indefinite noun phrases are represented without using existential quantifiers and quantification is taken care of by the larger context. It depends on this larger context whether an indefinite noun phrase gets an existential reading or not. The life span of a discourse referent depends on the way in which it was introduced. All "alive" referents may serve as antecedents for anaphors is subsequent discourse. Anaphoric pronouns are represented as free variables linked to appropriate antecedent variables. Definite descriptions in their simplest use are treated in a way which is similar to the treatment of anaphoric pronouns: definite noun phrases in their anaphoric use are treated like indefinite noun phrases, i.e. they are translated as free variables, but give rise to additional anaphoric links. The difference between indefinite noun phrases on one hand and definite noun phrases and pronouns on the other, is that indefinites introduce new variables, whereas the variables introduced by definites and pronouns always are linked to an already established context. In other words, the difference between definites (including pronouns) and indefinites is that the former refer to entities that have been introduced before, i.e. to familiar entities, while the latter do not. Quantifier determiners, i.e. determiners of noun phrases which are neither definite nor indefinite, can bind more than one variable. Specifically, they can bind a block of free variables some of which may have been introduced by indefinites. Conditional operators can also bind blocks of free variables. Not all variables introduced by indefinites are in the scope of a quantifier or a conditional operator. Those which are not are existentially quantified over by default. The processing of a piece of discourse is incremental. Each next sentence to be processed is dealt with in the context of a structure which results from processing the previous sentences. The processing rules decompose a sentence, replacing the various parts by conditions to be added to the structure.

DRT is fundamentally neutral about the computational mechanisms through which a pronoun refers to a particular entity in the discourse domain. It rather provides a general framework within which the output of intra- and inter-sentential anaphora resolution procedures contribute to the dynamically updated meaning of a text. One of the main objectives of this thesis is to support the view that Binding Theory only deals with a very specific subset of phenomena which are commonly considered as *anaphora resolution* instances. We underpin this position through computational arguments that advocate for a clear separation of tasks between different layers of human language faculty. According to this approach, Binding Theory (and therefore our algorithms) deals with the very basic and purely formal mechanisms which rule bound-variable interpretations of pronouns. We believe it is possible to fruitfully integrate our algorithmic procedure in order to filter out as soon as possible clearly incorrect readings, but a more thorough analysis of the interactions between our computational approach to Binding Theory and DRT issues are left for further development.

4

Towards an Integrated Computational Approach to Binding Theory

When I am working on a problem,
I never think about beauty.
But when I'm finished,
if the solution isn't beautiful,
I know it's wrong.

Buckminster Fuller, HOW, June 2003.

4.1 Introduction

Binding Theory has come a long way since its first formulation in the seminal works by Chomsky and Lasnik of the early 80's (see [12], [38], [39]). In the previous chapter we sketched three of the most influential interpretations that have been proposed for its principles, namely the classical coreferential interpretation (section 3.2), Reinhart's bound variable interpretation (section 3.3), and Reinhart and Reuland's Reflexivity approach (section 3.4), which marks a major depart from the traditional purely syntactic view of Binding Theory in favor of a more semantically oriented one.

In the present chapter we are going to explore how different approaches to Binding Theory issued from the last thirty years of linguistic enquiry may be effectively integrated into a computational framework. Our purpose is to enrich the current framework of computational semantics in order to inductively compute semantic representations of a sentence which incorporate the principles of Binding Theory.

Current approaches to formal semantics generally assume that the structures that feed the semantic interpretation module already come with the correct indexes associated to Determiner Phrases, where coindexing usually encodes sameness of reference. The implicit assumption is that Binding Theory rules a syntactic phenomenon, taken care of by a syntax module that generates structures suitable to be semantically interpreted, whose indexes associated to DPs conform to the principles of BT. Actually, very little is said about the computational procedure that assigns those indexes to DPs.

The original formulation of Binding Theory presents principles A, B and C as syntactic conditions that *indexed* Determiner Phrases must fulfill in order for the sentence in which they occur to be well-formed. Indexes are a formal device halfway between syntax and semantics that was introduced to encode coreferential relations between DPs in a sentence. Binding Theory principles are formulated in terms of syntactic binding, that is c-command plus co-indexing. They basically act as filters that discard every structure whose indexing violates any of A, B, or C principles. However, Determiner Phrases that occur in a phrase-marker issued from generative parsing of a sentence do not come with indexes associated. Principles A, B and C provide a procedure to *verify* that a given indexing for a sentence is BT-compliant, but they are *not constructive*: no effective procedure to associate correct indexing to DPs in a sentence is provided. This is both a theoretically and a practically challenging issue. How do human beings come to associate the correct indexing (i.e. to establish the correct mutual denotational relationships) to the DPs occurring in a sentence? And how can we devise a computational procedure to mimic this process in order to obtain a semantic representation for the sentence which encodes the additional information provided by the constraints of Binding Theory?

One may think that some internal language module computes all the possible combinations of indexes for the DPs and then filters out those which do not comply with any of A, B or C condition. But we have already seen (see section 3.6.5) that free indexation produces a number of indexings per phrase structure which is exponential in the number of DPs (see [18], [56]). Any naïve algorithm which computes free indexation for the DPs in a sentence and then filters the results through a module which verifies whether every indexing meets BT constraints would have to perform a number of checks exponential in the numbers of DPs. Such a fundamental result dismisses the brute-force combinatory approach as computationally impracticable, in addition to be highly implausible from a psycholinguistic point of view.

Our goal is to compute the correct (and only the correct) indexings for a sentence in an inductive, bottom-up fashion that parallels the computation of a semantic representation for a sentence according to the principles and methods described in chapter 2. This would make it possible to filter out incorrect indexings for fragments of a sentence at earlier stages of the semantic interpretation process. By enabling the computational system to assign the correct indexings to smaller fragments of sentences, such an approach opens the possibility to use some kind of underspecified semantics (see for example [17]) to work with partial semantics representations generated during the inductive process.

In this chapter we tackle the problem of integrating in a computational semantics framework the mechanisms needed to encode the principles of Binding Theory into the semantic representations computed of a sentence. Different interpretations that have been given to Binding Theory ask for different implementations of such mechanisms. Quite ambitiously, we believe that, besides empirical linguistic coverage, the notions of computational efficiency and algorithmic elegance can act as useful guidelines in the development of formal semantics systems that integrate the principles of Binding Theory, and provide an interesting perspective over one of the most actively debated topics in modern linguistics and formal se-

mantics. Eventually, we propose an integrated approach that incorporates some of the basic features of the approaches described into a framework which is both computationally effective and linguistically well-grounded.

This chapter is structured in a dynamic, trial-and-error way that mirrors the personal intellectual path that the author followed in departing from the first tentative algorithmic translation of the coreferential approach to BT to reach an original synthesis which integrates some of the major insights of theoretical linguistics into a computationally sound system. We believe that this style of exposition is effective in showing how computational shortcomings and algorithmic inelegance of each approach can act as a drive to move towards a more sophisticated linguistic perspective and, the other way round, how the need for computational efficiency and elegance can provide some insights into the more theoretical issues of Binding Theory, and ultimately linguistics as a whole.

In section 4.2 we introduce the basic notation and conventions we will use throughout the chapter. We start our investigation by sketching in section 4.3 an algorithm which implements the classical coreferential interpretation of Binding Theory, and we address the problem of implementing the notion of mandatory or forbidden coreferentiality within a computational framework. In section 4.4 we show how some computational shortcomings of the previous approach can be overcome by implementing an interpretation of BT principles inspired by Reinhart 1983 [50]. These two approaches share a basic flaw: the notion of binding domain remains elusive and potentially threatening from a computational point of view. This comes as a consequence of the fact that at the current stage of development Binding Theory hasn't yet come up with an inductive, compositional characterization for the notion of binding domain for a Determiner Phrase which is satisfactory both from an empirical and a computational point of view. We overcome this basic limitation in section 4.5, where we propose an integrated algorithmic approach to Binding Theory which merges features from Chomsky 1981 [11], Reinhart 1983 [50], and Reinhart and Reuland 1993 [54]. We believe this to be the first accomplished effort to integrate within a single coherent computational framework some of the basic achievements and insights in Binding Theory issued of the last 30 years of linguistics and formal semantics enquiry.

4.2 Basic Assumptions and Notation

The algorithms we present share some common basic elements: (a) the input is the phrase-marker (or parse tree) of a sentence issued from a generative analysis; (b) the phrase-marker can be possibly re-structured by cyclic applications of Quantifier Raising over DP constituents triggered by specific structural conditions detected during the interpretation process; (c) semantic representations are generated in the traditional bottom-up fashion described in chapter 2, enriched with additional formal machinery to encode in the final logical form the information collected during step (b).

We assume that the input parse trees are binary branching. As already discussed in section 2.1.1, although most of the algorithmic apparatus we present can be easily extended to other types of parse trees, we stick to this long-established as-

sumption which still lies at the core of the most recent developments in generative syntax (see [14]).

For a given sentence s, let τ_s be a generative parse tree for s and $\mathcal{N} = \{n_1, n_2, \ldots, n_q\}$ be the set of nodes in τ_s, each corresponding to a different syntactic constituent[1]. Let $\mathcal{D} \subseteq \mathcal{N}$ be the set of nodes in τ_s which correspond to Determiner Phrases (DPs), which is partitioned into three disjoint sets $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{R}$, whose members are the nodes corresponding to reflexive pronouns (or anaphors), (non-reflexive) pronouns and full-DPs (or r-expressions) respectively. We assume that we can always tell which of the sets $\mathcal{A}$, $\mathcal{P}$ or $\mathcal{R}$ a node $n \in \mathcal{D}$ belongs to, and that we have at our disposal the following functions and predicates:

Definition 4.1. *A binary predicate* local *is defined on* $\mathcal{D}\times\mathcal{D}$ *such that* $local(n_i, n_j) = TRUE$ *iff nodes* n_i *and* n_j *correspond to two DP constituents which belong to the same local domain.*

Definition 4.2. *A binary predicate* agr *is defined in* $\mathcal{D}\times\mathcal{D}$*, such that* $agr(n_i, n_j) = TRUE$ *iff the agreement features of DP constituents corresponding to* n_i *and* n_j *are mutually compatible.*

Definition 4.3. *A function* dps(n) *is defined from* $\mathcal{N}$ *to* $P(\mathcal{N})$*, which returns the set of nodes corresponding to DP constituents within the constituent rooted in* n*.*

Implementing function *local* is far from being trivial from a computational point of view, and it virtually incorporates a large part of what Binding Theory was originally considered to be about. For the time being we stick to a "black box" approach and we do not delve deeper into the inner workings of this function to concentrate on the overall structure of the algorithms.

Function *dps(n)* is easier to devise. Since the algorithms we present work in a bottom-up fashion, we can think of collecting all the DPs encountered during the bottom-up traversal of the subtree rooted in n (including n itself if it's a DP node) in a register which is always accessible to the computational system.

4.3 The Coreferential Approach

4.3.1 The coreferential interpretation of Binding Theory

The first algorithm we detail is inspired by the classical interpretation of Binding Theory, as was first stated in the pioneering works of Chomsky and Lasnik, that we presented in section 3.2 and that we shortly recall here. This interpretation is based on the notion of *coreferentiality*, or sameness of reference. Anaphoric elements such as non-demonstrative pronouns and reflexives are seen as linguistic items that lack intrinsic denotation or reference. The *antecedent* is the linguistic element from which the anaphor gets its reference, with which it is said to be *coreferential*. The semantic notion of coreference between two DPs is encoded in

[1] Where this does not generate confusion, we often blur the distinction between a node in the parse tree and the corresponding syntactic constituent, e.g. we say that node n *is* a Determiner Phrase.

the syntax by means of *coindexing*: coreferential DPs carry the same index, and DPs with the same index must have the same denotation.

The principles of Binding Theory in this interpretation are expressed as conditions on the syntactic well-formedness of indexed sentences, in which two DPs carrying the same index are supposed to have the same denotation. They focus on the notion of *syntactic binding* between DPs, which occurs when two DPs in a mutual relation of c-command are coindexed. Principle A, B and C rule the configurations in which different types of DPs must or must not be (syntactically) bound in a well-formed sentence. Since c-command relations are given in the structure of the phrase-marker that our algorithm takes as input, while coindexing is not, Binding Principles ultimately translate into as many conditions that rule when two DPs must or must not be coindexed, i.e. coreferential.

If we want to devise a computational procedure to infer all the correct indexings for the DPs in a sentence, we have to reverse the direction of application of Binding Theory principles. In a well-formed sentence Principles A, B and C must hold. Each principle deals with one particular class of DPs: principle A with reflexive pronouns, principle B with non-reflexive pronouns, principle C with full-DPs. That means that for any given DP only one of the three principles (which translate into as many constraints on its status - either bound or not -) applies. But by definition if DP_1 is bound by DP_2, then DP_1 must be c-commanded by and coindexed with DP_2. Therefore, if we can establish whether DP_1 is c-commanded by DP_2 or not, we can tell which kind of constraint the applied principle imposes on the index (and thus on the denotation) of DP_1. The implication scheme is the following:

$$\text{anaphor} + \text{local c-command} \overset{A}{\Longrightarrow} \text{mandatory coindexing}$$
$$\text{pronoun} + \text{local c-command} \overset{B}{\Longrightarrow} \text{forbidden coindexing}$$
$$\text{full-DP} + \text{c-command} \overset{C}{\Longrightarrow} \text{forbidden coindexing}$$

In order for a sentence like *John likes John* to be correct, principle C (a full-DP must not be bound) states that the first and the second occurrence of *John*, which are in a c-command relation, must not be coindexed, and thus they necessarily refer to two distinct individuals in the real world that happen to have the same name but different semantic content. Otherwise stated, $John_i$ *likes* $John_j$ is well-formed for each pair $(i, j) \in \mathbb{N} \times \mathbb{N}$ such that $i \neq j$ and therefore, since contra-indexing encodes denotational disjunction, $[\![John_i]\!] \neq [\![John_j]\!]$. Analogously, in the sentence *John likes him*, principle B (a pronoun must not be bound by another local DP) forces *John* and *him* (the first DP c-commanding the second) to refer to distinct entities in the real world. As in the previous case, $John_i$ *likes* him_j is considered well-formed for each pair (i, j) such that $i \neq j$, that is $[\![John_i]\!] \neq [\![him_j]\!]$. On the other hand principle A (a reflexive pronoun must be bound by a local DP) states that *John blames himself* is well-formed if and only if *John*, which c-commands *himself*, is also coindexed with it, that is $i = j$ and therefore $[\![John_i]\!] = [\![himself_j]\!]$. The coreferential interpretation of Binding Theory advocates a tight correspondence between linguistic entities and real world objects: principles A, B and C not only govern the distribution of DPs within a sentence, but they also constrain the way human beings refer to real world entities by means of in a human language. If equal indexes encode sameness of denotation for two DPs, computing the cor-

rect indexing(s) for the DPs that occur in a sentence amounts either to assign the same denotation to coreferential DPs, or to compute conditions that forbid that possibility, respectively.

According to the coreferential interpretation, principle A states that a reflexive pronoun *must* take its reference within its local domain. That is, it establishes a functional dependence between the reference of the reflexive pronoun and exactly one of the DPs which belong to its local domain. This functional dependence cannot but be some kind of identity function. If we consider the sentence

(1) John$_i$ blames himself$_j$

we expect the semantic interpretation procedure to fail every time $i \neq j$. This happens when there is no suitable antecedent for *himself* within its local domain, either because there's no c-commanding DP, or because there is one whose agreement features are incompatible with those of *himself*. In this case we want the interpretation procedure to fail, because the sentence is ill-formed and it must not be possible to assign any truth value to it. When a suitable antecedent exists, the two arguments of the predicate *blames* must end up referring to the same entity. In the case of (1) the logical form that we want to get as the semantic interpretation for the sentence is simply BLAME(JOHN, JOHN).

Principles B and C define conditions under which two distinct DPs *must not* have the same reference. It is just natural to translate them into formal devices that make the interpretation procedure fail whenever the denotations of the two DPs are identical, according to a notion of identity that will depend on the semantic domain into which syntax is mapped by interpretation. If we consider the sentence:

(2) John$_i$ blames him$_j$

we want to get a semantic representation which makes explicit the fact that *John* and *him* must have different denotations, that is $i \neq j$. We may encode this information in the resulting semantic representation of the sentence by enriching the formalism of our object language. Each logical form thus will have the following structure:

$$F(q_1, \ldots, q_n)|(q_i, q_j), \ldots (q_l, q_k)|$$

where the q_is are type e entities that occur in F. The semantic interpretation of such kind of formula is as follows:

$$[\![F(q_1, \ldots, q_n)|(q_i, q_j), \ldots (q_l, q_k)|]\!] = \begin{cases} [\![F(q_1, \ldots, q_n)]\!] & \text{if}([\![q_i]\!] \neq [\![q_j]\!]) \wedge \ldots \\ & \ldots \wedge ([\![q_l]\!] \neq [\![q_k]\!]) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Such kind of obviative constraints on the interpretation of a sentence seamlessly propagate throughout the basic operations of computational semantics like functional application, lambda abstraction and conjunction. If we indicate with $\mathcal{C}_F$ the set of constraints associated to a given formula F, we posit:

$$A|\mathcal{C}_A| \cdot B|\mathcal{C}_B| = A \cdot B|\mathcal{C}_A \cup \mathcal{C}_B|$$
$$A|\mathcal{C}_A| \wedge B|\mathcal{C}_B| = A \wedge B|\mathcal{C}_A \cup \mathcal{C}_B|$$
$$\lambda x.(A|\mathcal{C}_A|) = (\lambda x.A)|\mathcal{C}_A|$$

To sum it up, in order to implement in a computational framework this approach to Binding Theory, we have to enrich the process of semantic interpretation in two ways. On one hand, we need to translate mandatory coreference induced by principle A for two DPs into a formal mechanism that guarantees that the two DPs get the same denotation at the semantic representation level. On the other hand, to encode forbidden coreference conditions for two DPs induced by principles B and C, we must enrich the semantic representation with suitable conditions that make the interpretation fail whenever the two DPs get the same denotation.

4.3.2 Algorithm's Outline

First step: computing F and M relations.

In the coreferential interpretation of Binding Theory, principle A has a *positive*, or constructive, content. It states that a reflexive pronoun mandatorily gets the same reference as another Determiner Phrase which c-commands it from the same local domain. Principles B and C have an eminently *negative*, or obviative, character: they define structural configurations that forbid sameness of reference between two DPs which are in a c-command relation. In the case of principle B, the prohibition is limited to DPs that c-command a pronoun within its local domain. Beyond the border of the local domain, nothing is stated: a pronoun can corefer with any other DP, either c-commanding it or not. For principle C the interdiction regards full-DPs, and applies to any c-commanding DP, either within or outside the local domain.

We need to define two basic relations between the Determiner Phrases, in order to model the two situations that are ruled by the principles of BT according to the coreferential interpretation: mandatory and forbidden coreference. We may think that this kind of structural information is collected during a first preprocessing phase for the parse tree, and it asks for additional semantic machinery to be translated into genuine semantic representations during the interpretation process.

Definition 4.4. *A binary predicate M (for "mandatory") is defined in $\mathcal{D} \times \mathcal{D}$, such that $M(n_i, n_j) = TRUE$ iff:*

- $agr(n_i, n_j) = TRUE$;
- n_i *c-commands* n_j;
- $n_j \in \mathcal{A}$;
- $local(n_i, n_j) = TRUE$.

Relation M holds between two nodes in a parse tree which correspond to a reflexive pronoun and another DP which c-commands it, has compatible agreement features and belongs to the same local domain. As such, they correspond to two DPs constituents which, according to principle A, must have the same denotation.

Definition 4.5. *A binary predicate F (for "forbidden") is defined in $\mathcal{D} \times \mathcal{D}$, such that $F(n_i, n_j) = TRUE$ iff:*

- n_i *c-commands* n_j;
- *one of the two following situations holds:*

- $n_j \in \mathcal{R}$;
- $n_j \in \mathcal{P}$ *and* $local(n_i, n_j) = TRUE$;

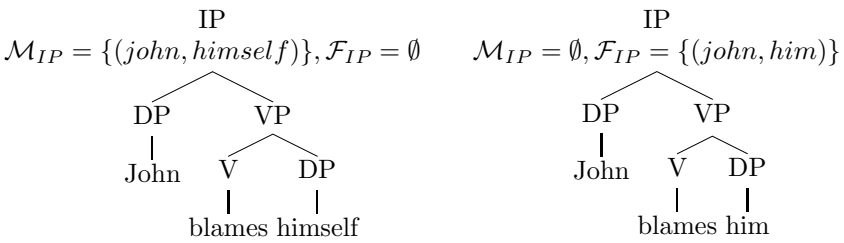
Relation F holds between two nodes in the parse tree which correspond to a DP which c-commands either a non-reflexive pronoun within its local domain, or a full-DP. As such, coreference between the two DPs is forbidden by principle B or C, respectively, and so their denotations are supposed to be different. Note that the condition on agreement is no longer necessary: if the two DPs do not have mutually compatible agreement features, *a fortiori* their denotations cannot be equal.

Each node n in the parse tree τ_s comes with two sets associated, $\mathcal{M}_n \subseteq \mathcal{D} \times \mathcal{D}$ and $\mathcal{F}_n \subseteq \mathcal{D} \times \mathcal{D}$. They can be inductively computed as follows (we recall from section 4.2 that $dps(n)$ is the function that returns the set of nodes corresponding to DPs occurring in the parse tree rooted at n):

- if n is a leaf, $\mathcal{M}_n = \mathcal{F}_n = \emptyset$;
- if n is a branching node, with n_1 and n_2 the two (immediate) daughters of n, then:
  1. if $n_1 \in \mathcal{D}$, for each $n_i \in dps(n_2)$:
     - if $n_i \in \mathcal{A}$ and $local(n_1, n_i) = TRUE$, then add (n_1, n_i) to $\mathcal{M}_n$;
     - if $n_i \in \mathcal{P}$ and $local(n_1, n_i) = TRUE$, then add (n_1, n_i) to $\mathcal{F}_n$;
     - if $n_i \in \mathcal{R}$, add (n_1, n_i) to $\mathcal{F}_n$;
  2. if $n_2 \in \mathcal{D}$, for each $n_j \in dps(n_1)$:
     - if $n_j \in \mathcal{A}$ and $local(n_2, n_j) = TRUE$, then add (n_2, n_j) to $\mathcal{M}_n$;
     - if $n_j \in \mathcal{P}$ and $local(n_2, n_j) = FALSE$, then add (n_2, n_j) to $\mathcal{F}_n$;
     - if $n_j \in \mathcal{R}$, add (n_2, n_j) to $\mathcal{F}_n$.

  Note that no assumption is made on the linear order of n_1 and n_2.

It can be easily verified that for each n in τ_s, with n_1 and n_2 its children nodes, sets $\mathcal{F}_n$ and $\mathcal{M}_n$ contain pairs of nodes for which relations F and M respectively get established when subtrees rooted at n_1 and n_2 merge into a tree rooted at n. Agreement and locality are verified by definition, while c-command relation is verified by construction: both point 1 and 2 take into account only (and all) couples (n_i, n_j) where n_j is either a sister or a descendant of a sister of n_i, which is one of the possible characterizations for the c-command relation, as seen in the previous chapter. We can then think that the structures which are fed to the semantic interpretation module look like the following parse trees for sentences *John blames himself* and *John blames him*, after this first "pre-processing" phase:



From relations to denotations.

Information collected during the previous step into sets $\mathcal{M}$ and $\mathcal{F}$ is used to enrich the interpretation procedure, that is the mapping from phrase-markers into semantic representations in our object language. In order to do that, we are going to add some sophistication to the inductive computational procedures described in section 2.2 to take into account, when computing the semantics of a node n, the information stored in sets $\mathcal{M}_n$ and $\mathcal{F}_n$.

We adhere to the common practice in semantics of translating pronouns, either reflexive or non-reflexive, as variables. We make sure by construction that each pronoun encountered in our bottom-up interpretation procedure is translated as a fresh new variable taken from an arbitrarily large supply: $x_1, x_2, \ldots, x_n$. This simple stipulation wired into the algorithm guarantees that, unless otherwise explicitly specified, neither mandatory nor forbidden coreference is stated between two DPs. This matches the fact that BT does not have anything to say about pairs of DPs which are not in a c-command relation, while mandatory and forbidden coreference must be stated explicitly. Relations M and F are the formal counterpart of the intuitive idea that coreference must be stated or forbidden between pairs of c-commanding DPs according to the principles of Binding Theory. The interpretation procedure must be enriched to translate the information stored in sets $\mathcal{M}$ and $\mathcal{F}$ into semantic representations.

By construction, set $\mathcal{M}$ contains pairs of DPs that must have the same denotation as a consequence of Principle A. We implement this condition by means of the well-known lambda calculus operation of *simple substitution*. For each pair $(n_i, n_j) \in \mathcal{M}$, we have by construction that $n_j \in \mathcal{A}$, so its semantic translation is a variable, say $[\![n_j]\!] = x_j$. Therefore the following translation rule is well defined:

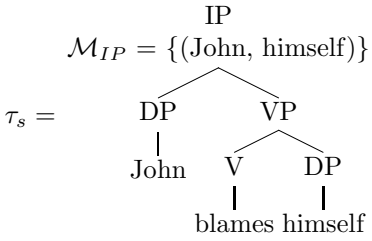
Interpretation rule for $\mathcal{M}$ set.

Let n be a node in a parse tree τ, with $\mathcal{M}_n$ associated, and S the logical form computed for n. If we assume $[\![n_j]\!] = x_j$, we define:

$$[\![n, \mathcal{M}_n]\!] = \bigcup_{(n_i,n_j)\in\mathcal{M}_n} \{S[[\![n_i]\!]/[\![n_j]\!]]\} = \bigcup_{(n_i,n_j)\in\mathcal{M}_n} \{S[[\![n_i]\!]/x_j]\}$$

Stated informally, the interpretation of a node in the parse tree with its set $\mathcal{M}$ associated is in general a *set* of semantic representations. Each element of the set is the result of the substitution in the logical form S computed so far of each occurrence of the variable corresponding to the anaphor with the denotation of the DP that c-commands it. Note that the former semantic representation S does not belong to the newly computed set. The assumption that each occurrence of a pronoun is mapped into a new variable, prevents variable capture side effects and the substitution is always "safe".

(3) s: John blames himself.



$$\Rightarrow \begin{aligned} [\![IP, \mathcal{M}_{IP}]\!] &= \{\textsc{blame}(\textsc{john}, x_1)[[\![John]\!]/[\![himself]\!]]\} \\ &= \{\textsc{blame}(\textsc{john}, x_1)[\textsc{john}/x_1]\} \\ &= \{\textsc{blame}(\textsc{john}, \textsc{john})\} \end{aligned}$$

When the interpretation process gets to compute the semantic representation for the IP node, $S = \textsc{blame}(\textsc{john}, x_1)$ has been computed as the corresponding logical form by means of traditional semantic computations. Such kind of representation is obviously incomplete and needs to be integrated with the information collected during the preprocessing phase in $\mathcal{M}_{IP} = \{(John, himself)\}$. The application of the interpretation rule for $\mathcal{M}_{IP}$ leads to a singleton set which contains a formula derived from S, in which the denotation of *John* has replaced every occurrence of the variable corresponding to the anaphor *himself*.

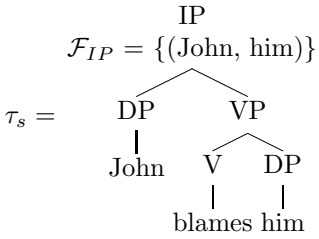
Set $\mathcal{F}$ contains pairs of DPs that according to principle B or C cannot be bound: since by construction they are in a c-command relation, they cannot be coindexed and thus they are supposed to have different denotations. In interpreting a node with a set $\mathcal{F}$ associated, we translate pairs in $\mathcal{F}$ into constraints in the formalism we introduced in section 4.3.1:

Interpretation rule for $\mathcal{F}$ set.

Let n be a node is a parse tree τ, with $\mathcal{F}_n$ associated, and S the logical form computed for n. We define:

$$[\![n, \mathcal{F}_n]\!] = \bigcup_{(n_i, n_j) \in \mathcal{F}_n} \{S|([\![n_i]\!], [\![n_j]\!])|\}$$

(4) s: John blames him.



$$\Rightarrow \begin{aligned} [\![IP, \mathcal{F}_{IP}]\!] &= \{\textsc{blame}(\textsc{john}, x_1) \mid ([\![John]\!], [\![him]\!]) \mid\} \\ &= \{\textsc{blame}(\textsc{john}, x_1) \mid (\textsc{john}, x_1) \mid\} \end{aligned}$$

In this case the preprocessing phase has computed a "forbidden coreference" relation induced by principle B between *John* and *him*, the latter being a non-reflexive pronoun locally c-commanded by the former. The relation is translated into a new semantic representation in which the logical form $S = \textsc{blame}(\textsc{john}, x_1)$, issued from standard semantic computations, is enriched with a condition that makes the computation fail whenever x_1 takes as its value the logical constant JOHN.

Second step: computing semantics.

Traditional computational semantics methods that we presented in chapter 2 compute a (unique) semantic representation for a node n in an inductive way: if n is a leaf its semantics is directly provided by the lexicon for the lexical entry associated; if n is a branching node, its semantics is computed through some basic operations (e.g. functional application, boolean conjunction, lambda abstraction) out of the semantic representations of its children nodes. As it is apparent from the previous step of the interpretation process, a sentence with anaphoric relations between its constituents in general allows multiple readings. Therefore, we need to modify this framework to compute *multiple* semantic representations associated to a given node, each of them corresponding to a different reading licensed by BT principles for the anaphors and pronouns involved. In order to do so, for each node n of the parse tree τ_s associated to a given sentence s, we compute a set $\mathcal{S}_n$ of lambda terms corresponding to the possible semantic representations associated. The information collected in sets $\mathcal{F}_n$ and $\mathcal{M}_n$ for each node enriches the semantic interpretation procedure in a way that takes into account the coreferential relations induced by principles A, B, and C according to the translation rules we introduced in the previous section.

Semantic interpretation for a sentence s takes as input the parse tree τ_s whose nodes are decorated with sets $\mathcal{M}$ and $\mathcal{F}$. It outputs a set of semantic representations $\mathcal{S}$ inductively computed for a generic subtree rooted at node n as follows:

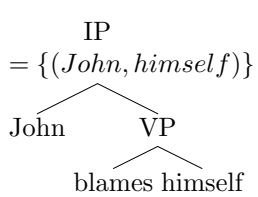
- if n is a leaf, then $\mathcal{S} = \{S\}$, where S is the lambda term which describes the semantics for the corresponding lexical entry as given in the lexicon;
- if n is a branching node, let n_1 and n_2 be its children, and $\mathcal{S}_1$ and $\mathcal{S}_2$ be the sets of semantic representations associated to each of them. Then for each $S_i \in \mathcal{S}_1$ and $S_j \in \mathcal{S}_2$, we initially set $\mathcal{S} = \bigcup_{i,j}\{S_i \cdot S_j\}$. For each $S \in \mathcal{S}$, let $\mathcal{M}_n$ and $\mathcal{F}_n$ be the sets associated to node n, then:
  - $\mathcal{S}' = \mathcal{S} - \{S\} \cup [\![(n, \mathcal{M}_n)]\!]$;
  - $\mathcal{S}'' = \mathcal{S} - \{S\} \cup [\![(n, \mathcal{F}_n)]\!]$.

  Resulting semantics is given by $\mathcal{S}' \cup \mathcal{S}''$.

Intuitively, for each semantic representation computed at a given stage of the interpretation process, if during the preprocessing phase a set of M or F relations between DPs occurring in the parse tree has been computed, we replace it with a reading in which mandatory or forbidden coreference conditions have been implemented through the translation rules introduced in the previous section.

4.3.3 Examples

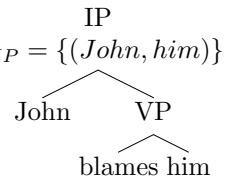
(5) s = John blames himself $\tau_s =$



Usual semantic computations described in chapter 2 yield the logical form $\textsc{blame}(\textsc{john}, x_1)$ for node IP. However, set $\mathcal{M}_{IP}$ computed during the pre-processing phase triggers the application of the translation rule for sets $\mathcal{M}$. The resulting semantics according to the previous section is thus:

$$\begin{aligned}[\![\tau_s]\!] &= [\![(IP, \mathcal{M}_{IP})]\!] = [\![(IP, \{(John, himself)\})]\!] = \\ &= \{\textsc{blame}(\textsc{john}, x_1)\} - \{\textsc{blame}(\textsc{john}, x_1)\} \cup \{\textsc{blame}(\textsc{john}, x_1)[\textsc{john}/x_1]\} = \\ &= \textsc{blame}(\textsc{john}, \textsc{john})\end{aligned}$$

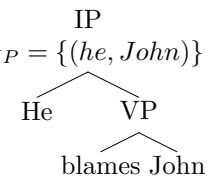
(6) s = John blames him $\quad \tau_s =$



Again, usual semantic computations generate the set of semantic representations $\mathcal{S} = \{\textsc{blame}(\textsc{john}, x_1)\}$ for node IP. In addition, pre-processing phase for the phrase-marker τ_s yields $\mathcal{F}_{IP} = \{(John, him)\}$. According to the translation rule for set $\mathcal{F}$ and the semantic interpretation procedure described in the previous section the resulting semantics is:

$$\begin{aligned}[\![\tau_s]\!] &= [\![(IP, \mathcal{F}_{IP})]\!] = [\![(IP, \{(john, him)\})]\!] = \\ &= \{\textsc{blame}(\textsc{john}, x_1)\} - \{\textsc{blame}(\textsc{john}, x_1)\} \cup \{\textsc{blame}(\textsc{john}, x_1)|([\![John]\!], [\![him]\!])|\} = \\ &= \{\textsc{blame}(\textsc{john}, x_1)|(\textsc{john}, x_1)|\}\end{aligned}$$

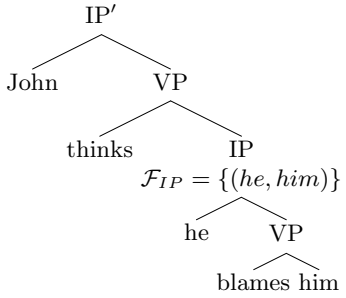
(7) He blames John. $\quad \tau_s =$



The situation is similar to the previous one, except that the forbidden coreference condition this time is triggered by an application of condition C.

$$\begin{aligned}[\![\tau_s]\!] &= [\![(IP, \mathcal{F}_{IP})]\!] = [\![(IP, \{(he, John)\})]\!] = \\ &= \{\textsc{blame}(x_1, \textsc{john})\} - \{\textsc{blame}(x_1, \textsc{john})\} \cup \{\textsc{blame}(x_1, \textsc{john})|([\![he]\!], [\![john]\!])|\} = \\ &= \{\textsc{blame}(x_1, \textsc{john}) \mid (x_1, \textsc{john}) \mid\}\end{aligned}$$

(8) s = John thinks that he blames him. $\quad \tau_s =$

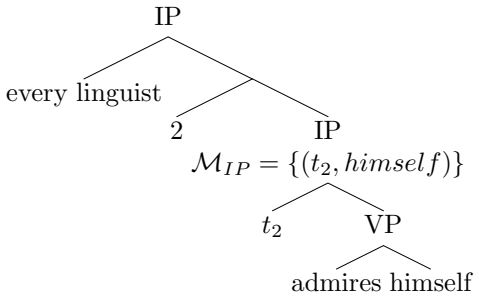


Since *John* c-commands the two pronouns *he* and *him* from outside their local domain, $\mathcal{F}_{IP}$ is the only set of relations between DPs that is computed for this parse tree, and *John* is free to corefer either with *he* or with *him* (but not with both of them at the same time), or with neither of them. The algorithm computes the semantic representation for the IP node analogously to what happens in example (6). If we assume $[\![he]\!] = x_2$ and $[\![him]\!] = x_1$: $[\![(IP, \mathcal{F}_{IP})]\!] = \{\text{BLAME}(x_2, x_1)|(x_2, x_1)|\}$. When computation reaches node IP' it performs the usual operations getting the semantic representation:

$$\begin{aligned}[\![\tau_s]\!] &= [\![VP]\!]([\![John]\!]) = \\ &= [\lambda x.\text{THINK}(x, \text{BLAME}(x_2, x_1)|(x_2, x_1)|)](\text{JOHN}) = \\ &= \text{THINK}(\text{JOHN}, \text{BLAME}(x_2, x_1))|(x_2, x_1)|\end{aligned}$$

In the final semantic representation, assignments $x_1 \rightarrow$ JOHN and $x_2 \rightarrow$ JOHN are both allowed but mutually exclusive: the constraint on their semantic value makes the interpretation fail whenever they're mapped into the same object, as desired.

(9) Every linguist admires himself $\tau_s =$



In order to correctly compute binding constraints for this sentence we make the (quite strong) assumption that our algorithm already operates on a parse tree which makes explicit the presence of a trace t_2 which is semantically bound by the quantificational DP *every linguist*. If we assume $[\![t_2]\!] = x_2$, the semantic representation for the fragment t_2 *admires himself* is, as in example (5), $\text{ADMIRE}(x_2, x_2)$. Therefore, the semantic representation computed for the whole structure is:

$$\begin{aligned}[\![\textit{every linguist}]\!](\lambda x_2.\text{ADMIRE}(x_2, x_2)) &= [\lambda P.\forall x(\text{LINGUIST}(x) \rightarrow P(x))](\lambda x_2.\text{ADMIRE}(x_2, x_2)) \\ &= \forall x.\text{LINGUIST}(x) \rightarrow \text{ADMIRE}(x, x)\end{aligned}$$

If we had *him* instead of *himself*, the semantic representation computed by our algorithm for the fragment t_2 *admires him* would be $\text{ADMIRE}(x_2, x_1)|(x_2, x_1)|$ and the overall semantics for the sentence:

$$\begin{aligned}[\lambda P.\forall x(\text{LINGUIST}(x) \rightarrow P(x))](\lambda x_2.\text{ADMIRE}(x_2, x_1)|(x_2, x_1)|) = \\ \forall x(\text{LINGUIST}(x) \rightarrow \text{ADMIRE}(x, x_1)|(x, x_1)|)\end{aligned}$$

4.3.4 Observations

As a first stab at a computational treatment of Binding Theory, we somewhat mechanically translated the ideas of coreferential interpretation into an algorithmic procedure which inductively computes suitable semantic representation. Our commitment to the coreferential approach of the early formulations of Binding Theory, which advocates a strong connection between the binding configurations and the denotations of linguistic entities, forced us to enrich the logical language of the final semantic representations with ad hoc machinery to deal with the obviative character of Principles B and C.

We have introduced two binary predicates, M and F, to keep track of the coreferential relations, induced by principles A, B and C, which hold between DPs occurring in a sentence: that is, mandatory coreference and forbidden coreference, respectively. It is apparent from its definition that F relation is very complicated, defined as it is on several ad hoc cases, which make it inelegant and very little intuitive. One gets the impression that we have tried to cover many heterogeneous phenomena with a single relation: phenomena that should be dealt with by different computation modules, we might suspect. The problem lies in forbidden coreference, which may come from different sources: condition B and condition C, which models very different phenomena. In a way this goes in the direction of much current research in linguistics, which tend to draw a neat separation between condition C and other conditions, the ultimate goal being condition C to disappear into some kind of general architectural principle. This will be made even more explicit in the next section.

Making "forbidden coreference" explicit forced us to invent a new formalism to keep track of and provide a suitable interpretation to such binding configurations. This sounds quite unnatural and cumbersome, both from a notational and a computational point of view. We'll see that the next approach we are going to adopt drops this idea as a whole: there is no such a thing as forbidden coreference, just like BT does not have anything to say about the relation between linguistic entities and their reference in the real world.

The net result of this first tentative computational treatment of Binding Theory according to its traditional interpretation is a somewhat heavy and unnatural design for the algorithm, which could be taken as a clue that we're mixing different levels of organization within a single module. Computationally, there seems to be a pressure to separate the task of Binding Theory from what other modules (discourse theory) are supposed to do. We believe that the approach we present in the next section goes in this direction.

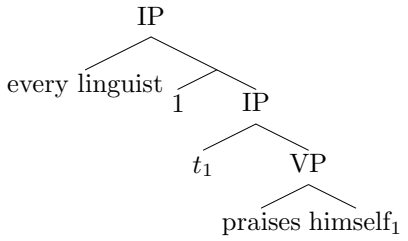
4.4 The Bound Variable Approach

4.4.1 Reinhart's interpretation of Binding Theory

The second algorithm we present is inspired by the interpretation of Binding Theory given by Tanya Reinhart in [50], which we detailed in section 3.3. According to this interpretation, Binding Theory does not deal with coreference relations between linguistic entities in a sentence, i.e. whether two distinct DPs can, must or must not refer to the same entity in the real world. Instead, Binding Theory only rules the conditions under which is it possible, impossible or mandatory to give a bound-variable semantic interpretation to two distinct Determiner Phrases, i.e. to have semantic binding between them. Coreferential readings are a byproduct of semantic binding, while the notion of forbidden coreference disappears in favor of a more general economy principle that favors interpretations which involve semantic binding over those which do not.

In section 3.3.1 we introduced the notion of semantic binding that we recall here. A DP α *semantically binds* a DP β if and only if β and the trace of α are semantically bound by the same variable binder, e.g. in (10) quantificational DP *every linguist* semantically binds *himself*, as encoded by coindexation between the pronoun and the trace:

(10)



The notion of semantic binding is central in the work of Reinhart [50] and was later systematized by Heim and Kratzer in [26]. Two semantically bound DPs always receive the same semantic interpretation, but this necessary coreferential effect must be kept conceptually distinct from other *accidental* coreference phenomena which are dealt with at pragmatic or discourse theory level. In a sentence like *John likes himself* what principle A of Binding Theory actually tells us is that *himself* and *John* must be *semantically* bound, that is they must be given a bound-variable interpretation. Therefore the semantic representation computed for (10) is $[\lambda x.\textsc{like}(x, x)](\textsc{john}) = \textsc{like}(\textsc{john}, \textsc{john})$. Coreference between *himself* and *John* is the byproduct of a purely formal property such as semantic binding, instead of a principled correspondence between linguistic entities and real world referents.

An important consequence of Reinhart's interpretation is that principle B does not forbid anymore sameness of reference between a non-reflexive pronoun and another DP, for the simple reason that according to this approach Binding Theory has nothing to say about the denotations of DPs in general. All that Principle B states is that a pronoun cannot be semantically bound with another DP which c-commands it within its local domain. Of course, that could very well happen when the c-commander lies outside the local domain of the c-commandee.

(11) a. John likes him.
b. John thinks that he likes Ann.

In (11-a), according to this alternative interpretation, principle B only forbids a semantic interpretation where *John* and *him* are semantically bound. In the language of formal semantics, this means that $[\lambda x.\textsc{like}(x, x)](\textsc{john})$ cannot be a semantic representation for (11-a). However, nothing prevents *John* and *him* from referring to the same individual in the real world. Indeed, we can think of conversational contexts in which this may be the case, like in the well known (although not uncontroversial) "Oscar sentences" (see [50]):

(12) It is not true that nobody likes John. *John* likes him!

The existence of contexts like (12) in which a sentence like *John likes him* is perfectly suitable to convey the intended meaning "John belongs to the set of individuals who like John", is considered by Reinhart as a proof that Binding Theory is perfectly neutral with respect to the semantic content of DPs in general, and non-reflexive pronouns in particular. Formally, this possibility is licensed by assigning to the sentence the logical form $\textsc{like}(\textsc{john}, x)$, where no semantic binding occurs between *John* and the pronoun *him*. The pronoun *him* is mapped as usual into a variable, which in the context of the sentence happens to be free. As such, no further constraint on the value it may take is stated. The context, as well as pragmatic or rhetoric factors, may very well map the free variable into any semantic entity, in particular $[\textsc{like}(\textsc{john}, x)]^{x \rightarrow \textsc{john}} = \textsc{like}(\textsc{john}, \textsc{john})$, which is the case of the last example. Coreferential relations are therefore a matter of discourse-theoretical strategies that could constrain the assignment of values to variables, and not the matter of Binding Theory, which only rules a very specific, internally grammatical property such as semantic binding. To sum up, semantic binding between two DPs implies sameness of reference; but lack of it in no case implies different denotations.

The optional character of principle B for pronouns c-commanded from outside their local domain is shown in example (11-b), where it licenses a bound variable reading between *John* and *he*. This means that we have a semantic representation for (11-b) as $[\lambda x.\textsc{think}(x, \textsc{like}(x, \textsc{ann}))](\textsc{john})$, in which *he* and *John* are bound by the same variable binder, which coexists with the logical form $[\lambda x.\textsc{think}(x, \textsc{like}(y, \textsc{ann}))](\textsc{john})$ in which no semantic binding occurs between *John* and *he*. An algorithmic implementation of Reinhart's approach must be able to generate both.

4.4.2 Reinhart's Principle I

At first sight, Reinhart's interpretation looks more attractive than the coreferential one from a computational point of view. As pointed out in section 4.3.4, "forbidden coreference" relation F asks for a case-by-case, ad hoc and eventually unnatural definition that accounts for two different configurations in which two DPs are supposed to have different semantic content. Furthermore, it asks for additional formal machinery at the interpretation level to keep track of the supposedly distinct denotations that must be assigned to DPs. Such algorithmic inelegance was taken

as a clue that maybe a rethinking of the phenomena we tried to grasp was needed. Now, in the light of Reinhart's interpretation of BT principles, cumbersome F relations between DPs disappear, turned into non-bound variable readings, which are provided by default by an algorithm which introduces a new variable at each occurrence of a pronoun or anaphor in the parse tree. A sentence like *John likes him* will be simply translated as $\textsc{like}(\textsc{john}, x)$, without additional machinery to encode the fact that *John* and *him* must have different denotations. Some important modifications of the algorithm are needed to account for optional or mandatory bound-variable readings when principle B or principle A apply, respectively.

It is not surprising that giving up to the idea of encoding in the logical form of a sentence constraints that are now considered as part of discourse strategies, the overall algorithmic structure of a "BT computational module" gets simplified. However, this superficial "unambitious" view of Reinhart's interpretation hides a major computationally challenging issue. As a matter of fact, a speaker who wants to convey the information that John likes himself, will utter the sentence *John likes himself* (where *himself* is translated into a bound variable later saturated by *John*) instead of *John likes him* (where the pronoun *him* is a free variable which can be mapped by the context into *John*). Consider the following logical forms:

(13) a. $\textsc{like}(\textsc{john}, \textsc{john})$
b. $[\lambda x.\textsc{like}(x, x)](\textsc{john})$
c. $[\textsc{like}(\textsc{john}, x)]^{x \rightarrow \textsc{john}}$

When a speaker wants to convey (13-a), he privileges a syntactic form whose direct semantic translation is (13-b) instead of one whose logical form is (13-c). Reinhart must account for the fact that although truth-conditionally equivalent, logical forms (13-b) and (13-c) are not interchangeable from the point of view the speaker. Therefore, she must distinguish them on the basis of another criterion. Reinhart's solution is "rule I" (also known as the Coreference Rule): if a given message can be conveyed by two minimally different logical forms of which one involves variable binding where the other has coreference, then the variable binding structure is always the preferred one. More formally:

Reinhart's rule I.

A DP α cannot corefer with a DP β if an indistinguishable interpretation can be generated by Quantifier Raising β and replacing α with a variable bound by the trace of β.

Although conceptually neat, this principle is computationally ruinous. Indeed, it implies being able to decide whether two distinct logical forms (conveyed messages corresponding to two different sentences) are truth-conditionally undistinguishable. That amounts to deciding whether they can be satisfied by the same sets of variable assignments. This problem is an instance of the well-known problem SAT[2], which is known from computational complexity theory to be NP-complete

[2] *Satisfiability Problem*(SAT) is the problem of finding a truth assignment that satisfies a given collection of Boolean clauses. The input is a set V of variables and a collection $C = \{C_1, C_2, \ldots, C_k\}$ of Boolean clauses over V. The output is a truth assignment that satisfies every clause in C. The corresponding decision problem is to decide if such a satisfying truth assignment exists.

(see [46] for a modern account of this as well as other important complexity theory results).

If computational intractability cannot be taken as an absolute criterion to reject a linguistic theory like Reinhart's, it severely undermines our undertaking to provide a straightforward implementation of Reinhart's interpretation of Binding Theory within a computational framework. However, Reinhart's interpretation marks an important departure from the previous approach and indirectly gives us clues to overcome some of the basic flaws that we outlined in section 4.3.4. The output of an hypothetical "Binding Theory module" is not anymore a set of exotic semantic representations enriched with constraints on the denotations of their DPs induced by the obviative effect of principles B and C, but a set of standard logical forms that describe all the possible semantic readings compatible with the sentence. Principle B and C will only act as criterions to exclude bound-variable readings under certain configurations. The focus of the algorithm thus lies in devising a mechanical procedure to generate such readings with the smallest number of assumptions on the nature of DPs. The problem raised by the implementation of Rule I is put on hold until section 4.5.4, where we'll provide a suitable computational treatment of this issue.

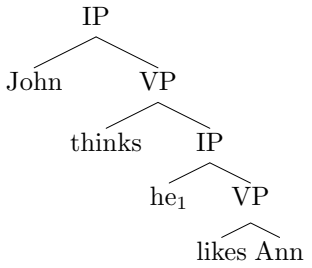
4.4.3 Binding principles revisited

Principle B

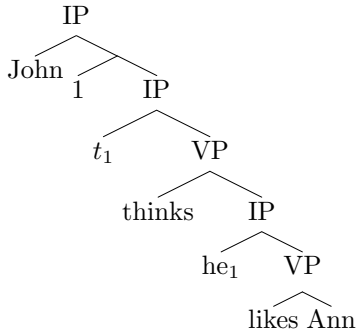
According to Reinhart's interpretation, principle B states that a non-reflexive pronoun cannot be *semantically bound* within its local domain. We can rephrase it by stating that a non-reflexive pronoun must occur free within its local domain, while it can occur either free or bound outside. This is the case of non-reflexive pronoun *he* in the following example:

(14) a. John thinks that he likes Ann.
b. $\textsc{think}(\textsc{john}, \textsc{like}(y, \textsc{ann}))$
c. $[\lambda x.\textsc{think}(x, \textsc{like}(x, \textsc{ann}))](\textsc{john})$

$\tau_1 =$



$\tau_2 =$



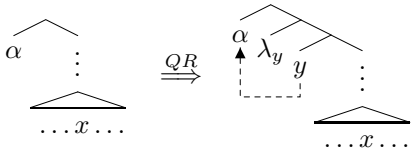
Logical form (14-b) results from the standard semantic interpretation procedure applied to τ_1. It corresponds to a reading in which the pronoun *he* is not bound, neither within nor outside its local domain, as is the corresponding variable in

the logical form. On the other hand (14-c) is another logical form compatible with (14-a). It is the result of the interpretation of τ_2, in which *John* has been Quantifier Raised, and where its trace t_1 and he_1 are bound by the same lambda operator, thus making *John* and *he* semantically bound. Accordingly, the variables which correspond to their semantic interpretation occur *bound* (in the logical sense) in (14-c). In particular, they are *two occurrences of the same variable* which are bound by the same lambda operator: one which corresponds to the c-commanded pronoun, and one occurring at the position corresponding to *John* in the phrase-marker. In order to encode principle B in a computational framework, we must be able to generate both readings for sentence (14-a).

Semantic representation (14-b) easily comes "for free" by introducing a fresh new variable as the semantic interpretation of each occurrence of a pronoun, either reflexive or not. If in no case in the logical form of a sentence there are two occurrences of the same variable, semantic binding (which requires *two* occurrences of the same variable to be bound by the same lambda operator) is by definition impossible. This implements the *obviative* side of principle B, by generating by default a logical form in which semantic binding is impossible for all mutual c-commanding DPs in a sentence.

Generating (14-c) is more complicated: we need a semantic representation in which *the same variable* occurs (a) at the position occupied by the c-commanded pronoun and (b) at a position of the c-commander DP; besides, (c) both variables must be bound by the same lambda operator. If we start from (14-b), we know that by construction all variables occurring in the logical form are different. As we will formally detail in section 4.4.5, we postulate that the *only* way to identify two DPs which are in a relation of c-command is to apply a slightly modified version of QR on the c-commander. This is part of our view (that we sketched at the end of section 2.2.7) of QR as a *last resort* device that is not freely and optionally available, but which is triggered under the pressure of specific circumstances: either semantic type mismatch or (as in this case) by the need to generate a bound variable reading between a pronoun and another DP in a c-commanding position.

We already know from the definition in section 4.4.1 that semantic binding at the logical form level is a formal device that exclusively deals with *variables* (it actually requires two occurrences of the same variable to be bound by the same lambda operator). However, in generating a bound-variable reading for a sentence like (14-a) we need to establish semantic binding between a full-DP (*John*, c-commander) and a non-reflexive pronoun (*he*, c-commandee). Only the latter can be directly mapped into a variable. QR is thus our last resort device to make a variable appear in the position previously occupied by the full-DP. We intuitively illustrate this procedure with the following schema:

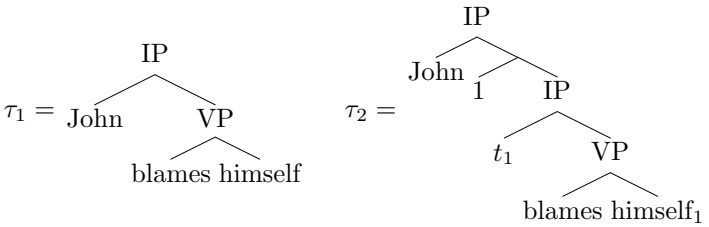


Semantic binding can be then achieved by identifying variables x and y.

Principle A

Principle A, rephrased in Reinhart's interpretation, states that a reflexive pronoun must be semantically bound within its local domain. We know from the definition of semantic binding that this means that there must be another DP within the same local domain such that the reflexive pronoun and the trace of the DP which has been Quantifier Raised are bound by the same lambda operator. This is the case of *John* and *himself* in the following example:

(15) a. John blames himself
b. *$\textsc{blame}(\textsc{john}, x)$
c. $[\lambda x.\textsc{blame}(x, x)](\textsc{john})$



In (15-c), issued from the semantic interpretation of τ_2, the variable which represents the semantic translation of the reflexive predicate *himself* and the variable corresponding to the trace left by *John* after QR are bound by the same lambda operator: therefore principle A is met and (15-c) is a correct semantic reading for (15-a). In (15-b) no semantic binding occurs between the two arguments of predicate *blames*, therefore principle A is not met and (15-b) is not acceptable as the logical form of (15-a).

As a corollary of Principle A we may say, blurring the distinction between pronouns and variables, that no reflexive pronoun can occur free (that is, unbound) within its local domain: **Ann thinks that John likes herself* must be considered ill-formed because *herself* is not bound within its local domain (and so is the corresponding variable at logical form level), which is the clause *John likes herself*.

The status of reflexive pronouns appears to be twofold. On the one hand they behave like pronouns as for semantic type and agreement features. On the other hand, as stated by principle A, they must be bound within their local domain, otherwise the sentence is ill-formed. This is a first important difference between logical variables and natural language pronouns: the latter come in two different forms, that is reflexive and non-reflexive. Such a distinction has no direct correspondence in the domain of logical variables. In order to mark this fundamental difference between reflexive and non-reflexive pronouns, we will adopt the convention of mapping reflexive pronouns into "starred" variables $x_1^*, x_2^*, \ldots, x_n^*$, and non-reflexive ones into "normal" variables.

It is clear that we can compute reading (15-c) for sentence (15-a) via an application of QR in exactly the same way we use QR to generate (14-c) as a logical form for (14-a): the only difference is that in this case this is the only correct reading we want to generate for the sentence. In (15-a) we are basically in the same DP configuration of (14-c): a (reflexive) pronoun (*himself*) is c-commanded by a

full-DP (*John*) with which we want to generate a bound variable reading. Here again QR is triggered by the need to (temporarily) replace the full-DP *John* with a variable that can be identified with the variable corresponding to the reflexive pronoun which get bound by the same lambda operator.

Principle C

Reinhart's interpretation of Binding Theory gets rid of condition C, at least in the explicit form of a third principle, partly because the shift from coreferentiality to semantic binding makes acceptable (within particular contexts - see the discussion on Oscar sentences) a whole class of interpretations formerly forbidden, partly because it's subsumed by more general principles like Rule I. This entails an important simplification of the formal apparatus we had to put in place to capture the three principles in the previous approach. Consider the following example:

(16) a. John likes John
b. $\textsc{like}(\textsc{john}, \textsc{john})$
c. $*[\lambda x.\textsc{like}(x, x)](\textsc{john})$

According to Reinhart's interpretation of BT, principle C now simply states that a full-DP cannot be semantically bound by another DP. That is, only (16-b) is the correct semantic interpretation for (16-a), while (16-c), although truth-conditionally equivalent to (16-b), is not. Once we adhere to the idea that BT does not deal with coreference relations, we do not need additional machinery to specify that in (16-a) the two occurrences of *John* must refer to distinct individuals in the real world. This means that we do not have to restructure our parse tree when a full-DP occurs in c-commanded position, and that makes for a significant simplification of the application of QR. What is particularly interesting is that this follows naturally from the use of QR we advocated so far, without any further stipulation. Let's make this point more precise.

The cases of semantic readings generated by the application of principle B and principle A, like in (14-a) and (15-a) respectively, share an important point: in both cases the c-commanded DP is a pronoun, reflexive in the case of applications of principle A, non-reflexive in the case of principle B. Both reflexive and non-reflexive pronouns are semantically translated into variables, and variables lie at the core of the definition of semantic binding, which is the focus of Reinhart's interpretation of Binding Theory. We have seen that whenever the semantic interpretation detects a variable which is c-commanded by another DP, we want the algorithm that implements this approach to BT to generate a bound variable reading between them. The only way to achieve that is through QR of the c-commanding DP, which leaves a trace (semantically, a variable) in the position previously occupied by the DP. Therefore, in our computational system QR is a formal device which is triggered by the need to identify two entities, the c-commanded pronoun (semantically a variable) and the c-commanding DP. Since the latter in general is not semantically a variable, QR kicks in to put a variable in its position without losing the information encoded in the DP.

In the case of principle C we are in a different situation. By definition, the c-commanded DP is neither an anaphor nor a pronoun, so its denotation *is not* a

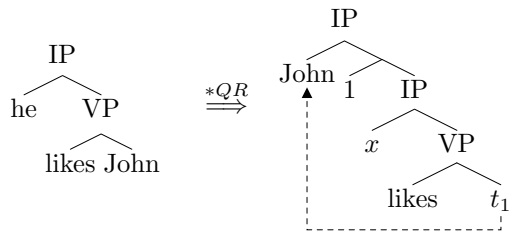
variable. We may ask why Quantifier Raising does not apply to the two occurrences of John in (16-a), thus getting (16-c) as the logical form for the sentence. But the fact is that there is no need for QR to apply here. Since the c-commanded DP is not (semantically) a variable there is simply no reason why QR, which is our device to generate semantic binding *between variables* gets triggered.

There is another configuration in which principle C must apply to exclude bound variable reading:

(17) a. He likes John
b. $\textsc{like}(x, \textsc{john})$
c. $^*[\lambda x.\textsc{like}(x, x)](\textsc{john})$

Even though the context may license an interpretation in which *he* denotes John (like in Oscar sentences), the correct semantic reading for (17-a) is (17-b) and cannot be (17-c). We have said that it's the presence of pronouns (semantically, variables) that triggers the application of QR in order to generate semantic binding between two c-commanding DP. In (17-a) we have a pronoun in a c-commanding position, so one may think that this should trigger QR of the c-commanded *John* in order to generate a bound variable reading. If we stick to the idea that QR gets triggered to replace a DP with a variable, it seems like there's no reason to forbid such kind of operation in a configuration like (17-a):

(18)



However, we believe that this possibility is forbidden by a constraint naturally imposed by the bottom-up structure of the procedure that we want to implement. That is, QR over a given DP α can be applied when and only when the interpretation procedure is processing α. Once the interpretation procedure has processed a node in the phrase-marker, it cannot backtrack and "undo" what previously was done by applying Quantifier Raising to a lower node in the tree. In (14-c) and (15-c) this is exactly what happens: when we get to process the higher DP node the interpretation algorithm detects a DP α which c-commands a *variable*, and it immediately QR the current DP node α to generate a bound variable semantic reading between the two DPs. In (17-a) *John* has already been semantically processed when the algorithm reaches bottom-up *he* and detects a c-command configuration between the two DPs. So the "QR cyclicity" constraint prevents the algorithm to apply QR to *John*, and thus getting the incorrect semantic reading (17-c). Principle C that previously required a cumbersome, *ad hoc* treatment, gets very naturally absorbed into a general architectural principle of our interpretation algorithm. That said, the issue raised by Reinhart's Rule I is still present: in general, since the same semantic content can be conveyed via a bound-variable interpretation, this latter should be privileged, and thus the expression *John likes*

himself. We will overcome this basic flaw of the present implementation in section 4.5.

4.4.4 On pronouns, variables and binding

Embracing Reinhart's alternative approach to Binding Theory, enriched with the observations made in the previous section, opens up the possibility to get rid of most of the cumbersome algorithmic apparatus introduced in section 4.3 to stick to a closer correspondence between natural language semantics, even enriched with the principles of Binding Theory, and lambda calculus. The very same notion of semantic binding of a pronoun closely reminds of the notion of variable binding in first order logic. We try to push this analogy as far as possible, and exploit it to implement in a inductive way the principles of Binding Theory according to Reinhart's approach.

In section 4.4.3 we have seen that semantic binding is a matter or (free) variables getting bound by the same lambda operator. A free (that is, unbound) pronoun in syntax gets translated at the logical level into a (logically) free variable. In lambda calculus, for any given lambda term τ, the set of free variables occurring in τ is an ubiquitous notion, and something that can be easily computed inductively. So, if we assume that each occurrence of a free variable in the lambda term that is the semantic representation of a syntactic constituent C is the semantic translation of an unbound pronoun (or trace) occurring in C, we have a very convenient way to inductively implement the revised principles of Binding Theory, getting rid of the artificial relations F and M needed to implement the coreferential approach in section 3.2.

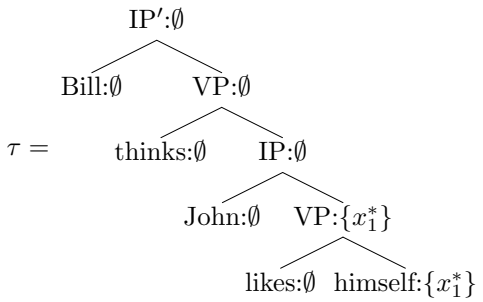
We can think that each node n in the parse tree comes with a set $\mathrm{FV}(n)$ of the variables that occur free in the lambda term that describes the semantics representation of n. This set can be inductively computed as follows:

- if n is a leaf and $[\![n]\!] = c$, where c is a constant, then $\mathrm{FV}(n) = \emptyset$;
- if n is a leaf and $[\![n]\!] = x_i$ for some variable x_i, then $\mathrm{FV}(n) = \{x_i\}$;
- if $n = n_1 \cdot n_2$, then $\mathrm{FV}(n) = \mathrm{FV}(n_1) \cup \mathrm{FV}(n_2)$
- if $n = \lambda x.n_1$, then $\mathrm{FV}(n) = \mathrm{FV}(n_1) - \{x\}$
- if $n = n_1 \wedge n_2$, then $\mathrm{FV}(n) = \mathrm{FV}(n_1) \cup \mathrm{FV}(n_2)$

From section 4.3.2, we know that, in processing in a bottom-up fashion a phrase-marker, when a type e DP A (that is, an entity) semantically merges with another constituent B, new c-command relations get established between A and any other type e DP occurring in the phrase-marker rooted at B. This is the only configuration in which new semantic readings can be generated. Let τ_A and τ_B be the logical forms of A and B respectively. Since we assume τ_A to have semantic type e, the only way to semantically combine it with another semantic term is as an argument applied via functional application to a type $e \to T$ lambda term, with T an arbitrary semantic type. So τ_B cannot but have type $e \to T$. We claim that all we need to generate all the semantic readings induced by the application of Binding Principles (modulo local domain considerations), is to generate, for each $x_i \in \mathrm{FV}(\tau_B)$, the corresponding semantic interpretation in which τ_A and x_i are bound by the same lambda operator. This approach presents several advantages:

No need for F and M relations. As we have extensively discussed in the previous section, QR must be triggered whenever a pronoun is c-commanded by another type e DP. Pronouns correspond to variables, therefore this corresponds to trigger QR for the c-commanding DP for each variable occurring in the set of free variables of the functor. The set of free variables of a lambda term evolves dynamically as the lambda term merges with others, therefore the set $\mathrm{FV}(n)$ keeps track of the interesting entities with which to create bound variable readings without additional machinery. Keeping track of how sets $\mathcal{F}$ and $\mathcal{M}$ associated to phrase-markers evolve as two of them get merged required a computational overhead which is not longer necessary. This is only more valid in the present approach where QR basically restructures the phrase-marker, and so we need a natural way to keep track of the changes in the c-command relations between DPs. Relations M and F become unnecessary and obsolete in the present approach, replaced by a single set of variables occurring free in the corresponding logical form.

Invisibility of bound entities. There's a hidden, yet meaningful assumption behind our choice of *free variables* as the only interesting c-commanded entities, which happens to be both linguistically sound and computationally convenient. Free variables correspond to unbound pronouns in the syntactic structure. This means that, in a bottom-up processing perspective, once a pronoun (either reflexive or not) gets bound by another DP, it is not available anymore to enter semantic binding with another DP. This eliminates a lot of redundancy which was implicit in the previous approach. Consider, for example, the following phrase-marker (where we label each node with the set of free variables in the corresponding logical form and we assume that $[\![himself]\!] = x_1^*$):



Our bottom-up interpretation procedure computes a bound-variable reading between *John* and *himself* as the only correct one for the fragment *John likes himself*. The set of free variables of the corresponding logical form $[\lambda x.\textsc{like}(x,x)](\textsc{john}) = \textsc{like}(\textsc{john},\textsc{john})$ is empty (the formula is closed). So, when the interpretation process reaches IP′ node, no interesting c-command relation is detected between *Bill* and a DP occurring in the VP subtree. This is due to the fact that *himself* is bound to *John* and cannot enter any semantic binding relation anymore. As soon as *himself* gets bound in the inner clause, it literally "disappears" from the scope of *Bill*. In particular, if a reflexive pronoun occur free from outside its local domain, and so a starred variable occurs in the $\mathrm{FV}(n)$ for a given node n which contains its local domain, the sentence is ill-formed and the interpretation procedure fails. If

the sentence is well-formed, and therefore meets Principle A, a reflexive pronoun gets bound within its local domain, and then the corresponding variable won't occur in the set of free variables that will be "visible" to a new DP that c-commands it from outside its local domain. This is in accord with the principles of Binding Theory and matches hygiene conditions for the substitution of variables in first order formulae. The set of free variables associated to node IP faithfully reflects this. In the coreferential approach, additional redundant relations F would have to be computed between *himself* and any other DP c-commanding them until the end of the computation, no matter how deeply embedded in the phrase-marker, and would ultimately result in several redundant "forbidden coreference conditions" in the final semantic representation. All this useless computational overhead gracefully disappears by adopting this approach. This was not the case for previous approach, for which it was necessary to stipulate that a DP c-commanding a reflexive from outside its local domain didn't perform any operation on it.

Natural implementation of Principle C. This approach smoothly implements the conclusions issued from the discussion in section 4.4.3 about principle C. We know that full-DPs, when c-commanded, do not trigger any QR operation. But this is exactly what happens if we limit our attention to *free variables* in τ_B when it functionally applies to type e τ_A. Since a full-DP is never interpreted as a variable this amounts to say that they become totally invisible to other DPs c-commanding them because they'll never enter the set $\mathrm{FV}(n)$. The only c-command relations that are of interest to this refined interpretation process are those which get established between a type e DP τ_A and each of the variables occurring in $\mathrm{FV}(\tau_B)$, that is the set of free pronouns, both reflexive or not, and traces.

From c-command to functional application. From the treatment of binding principles we have sketched so far, we get a new perspective on the nature and role of the c-command relation. In the original formulation of Binding Theory in the coreferential framework, c-command as a structural necessary condition for binding comes as a stipulation. The treatment of Binding Principles that we have sketched replaces such a stipulative structural approach with a more naturally semantic one. C-command is not explicitly stated anymore as a condition on binding, but it simply derives from the overall structure of the algorithm. We're not looking for c-command relations anymore, at least explicitly, but for relations that get established when a type e DP occurs as the argument of a predicate which contains other "free entities". This point will be further developed in section 4.5.

An important point where the correspondence between variables in logical formulae and pronouns in natural language sentences falls short is given by the notion of binding domain for the latter. There is no such a thing as a binding domain in first order logical formulae (essentially, because logical binding is not a 2-places relation as is semantic binding). For the time being we stick to the assumption made in section 4.2 that we have a black box predicate $local(\tau, n_1, n_2)$ which returns TRUE if nodes n_1 and n_2 belong to the same local domain in τ, and FALSE otherwise.

4.4.5 Algorithm's Outline

The interpretation algorithm takes as input a generative parse tree τ_s associated to a given sentence s. It returns a set $\mathcal{S}$ of semantic representations, each corresponding to a possible reading for the sentence according to the interpretation of Binding Theory principles given by Reinhart and detailed in sections 3.3 and 4.4.

The algorithm integrates the usual computations involved in the semantic interpretation process that we presented in chapter 2. It inductively proceeds bottom-up, by computing the logical form of a branching node n from the semantic representations of its children nodes n_1 and n_2. Logical forms of the leaves are directly provided in the lexicon for non-pronoun constituents. There are two basic steps at the core of the new interpretation process:

- each occurrence of a pronoun in the phrase-marker is translated into a fresh new variable; non-reflexive pronouns are translated as standard variables identified by the usual letters x_i, reflexive pronouns as "starred" variables x^*_k;
- as the semantic interpretation proceeds bottom-up, QR may be triggered to generate a semantic binding between two type e DPs occurring in the phrase-marker, therefore restructuring it.

From section 4.4.3 we know that the interpretation process generates bound variable readings whenever it's possible. In order to do that the algorithm starts by assigning different variables to every occurrence of a pronoun (thus de facto making any semantic binding impossible), and then when it detects a configuration that asks for an application of principle A or principle B it generates a bound variable reading by means of QR. Technical implementation details follow in the next section.

Rearranging the parse tree

The core of the interpretation algorithm lies in additional semantic computations that parallel the usual process of semantic interpretation as described in chapter 2. These computations are performed on the parse tree in correspondence to a specific configuration, that is when a type e DP A is combined with a (n-ary) predicate B. From a semantic point of view, that means that $\tau_B^{e\to T}$ gets applied by functional application to τ_A^e. Since τ_A is of type e it cannot but act as the argument, and τ_B as the functor. New c-command relations between type e DP constituents that get established at the stage can trigger a restructuring of the parse tree by means of applications of Quantifier Raising (for an explanation of why QR applies at this point, see our discussion in section 4.4.3).

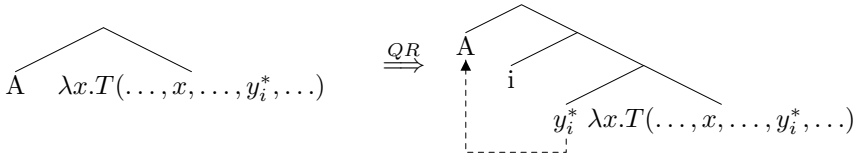
When τ_B gets applied to τ_A by means of functional application, new c-command relations get established between τ_A (which corresponds to the c-commanding DP) and every variable occurring free in τ_B (corresponding to the c-commanded free pronouns or still unbound traces). More explicitly, when we have $A^e \cdot B^{e\to\tau} = A^e \cdot \lambda x^e.T(x)^\tau$, for each $z \in \mathrm{FV}(B) = \mathrm{FV}(T) - \{x\}$, we distinguish the following situations:

1. z is a starred free variable, that is $z = y^*_i$ and therefore $B = \lambda x.T(\ldots, x, \ldots, y^*_i, \ldots)$. Two further subcases are possible:

a) $local(\tau, A, y_i^*) = TRUE$. We apply QR to A within the parse tree, and we copy y_i^* in the position. Otherwise stated, we create a new parse tree in which the term $A \cdot B$ is replaced in the following way:

$$A{\cdot}B = [\lambda x.T(\ldots, x, \ldots, y_i^*, \ldots)](A) \stackrel{QR}{\Longrightarrow} [\lambda y_i^*.[[\lambda x.T(\ldots, x, \ldots, y_i^*, \ldots)](y_i^*)]](A)$$

A is Quantifier Raised and it leaves a trace whose semantic interpretation is a variable which is *a copy* of the free variable that triggered the QR, in this case y_i^*. In the notation of phrase-markers[3]:



b) $local(\tau, A, y_i^*) = FALSE$. The computation fails.

2. z is a normal free variable, that is $z = y_j$ and $B = \lambda x.T(\ldots, x, \ldots, y_j, \ldots)$.
   a) $local(\tau, A, y_j) = TRUE$. The algorithm does not do anything: principle B forbids bound variable interpretation, and this is trivially implemented by default different variables assigned to different pronouns (semantic binding requires at least two variables occurring in a logical form to be equal, and bound by the same lambda operator).
   b) $local(\tau, A, y_j) = FALSE$. In addition to the default non-bound variable reading provided by default by the algorithm we add a new reading in which parse tree τ has been replaced by quantifier raising like in the previous case.

Let's analyze each of the four cases in detail.

Case 1a corresponds to a configuration in which the type e DP A c-commands a reflexive pronoun, whose semantic interpretation is the (for now) free variable y_i^*, and they both belong to the same local domain. According to principle A, in this configuration there must be semantic binding between τ_A and y_i^*. From section 4.4.3, we know that to get this reading we have to replace the DP with a variable, identify it with the variable corresponding to the c-commanded pronoun, and bind both of them by means of the same lambda operator. The only way to achieve that is by restructuring the parse tree via an application of QR over A. A is QR'd to a higher position in the tree, leaving a trace which is mapped into the variable y_i^*. The outcome of this operation is that the same variable now occurs in two different positions within B, bound by the same lambda operator, which is precisely the definition of semantic binding that we wanted to implement in this case:

$$\begin{aligned} [\lambda x.T(\ldots, x, \ldots, y_i^*, \ldots)](A) &\stackrel{QR}{\Longrightarrow} [\lambda y_i^*.[[\lambda x.T(\ldots, x, \ldots, y_i^*, \ldots)](y_i^*)]](A) = \\ (\beta\text{-reduction}) &= [\lambda y_i^*.T(\ldots, y_i^*, \ldots, y_i^*, \ldots)](A) = \\ (\beta\text{-reduction}) &= T(\ldots, A, \ldots, A, \ldots) \end{aligned}$$

[3] For sake of clarity and where this does not generate confusion, we often mix the denotation of phrase-markers with the semantic representations of their nodes.

In our system, QR is strictly a local movement: the landing site must necessarily fall immediately before the boundary of the local clause. This assumption, not uncontroversial in linguistics, is relatively well grounded from an empirical point of view (see [57]).

Case 1b models the situation in which the type e DP A c-commands a reflexive pronoun but they do not belong to the same local domain. From the discussion of section 4.4.4, we know that if the starred variable y_i^* belongs to the set FV(B), but A does not belong to the same local domain as the reflexive pronoun whose interpretation is y_i^*, it means that y_i^* didn't get bound within its local clause (we recall that as soon as a pronoun gets bound its denotation disappears from the set of free variables associated to the logical form of the phrase-marker). This is a violation of principle A that makes the computation fail.

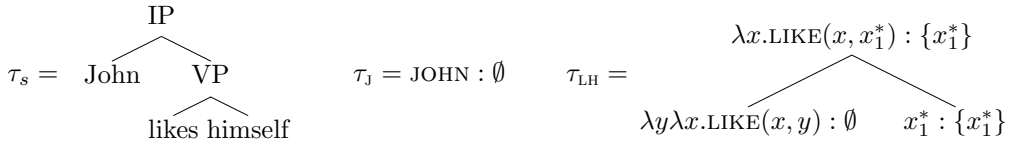
In 2a we have a type e DP A c-commanding a non-reflexive pronoun from within the same local domain. According to principle B, there cannot be a bound-variable reading between the two DPs. This is trivially implemented by the fact that the algorithm assigns a fresh new variable to each occurrence of a pronoun. Unless otherwise stated, no binding can occur between two distinct variables, and the algorithm does not perform any special operation, the only correct reading being the one without semantic binding between A and the non-reflexive pronoun.

In configuration 2b a type e DP c-commands a non-reflexive pronoun from outside its local domain. In Reinhart's interpretation of BT, principle B in this case licenses both free and bound variable readings between the two DPs. The former is provided by default by assigning a new variable to each occurrence of a pronoun, as in case 2a. The latter is generated by means of the same QR mechanism described in 1a.

4.4.6 Basic Examples

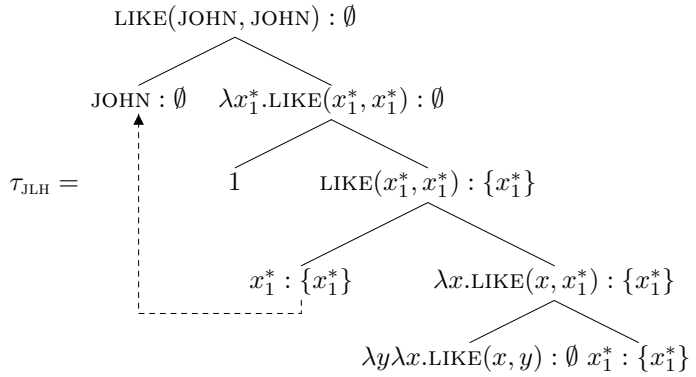
We provide some examples of how the algorithm works relatively to some basic configurations. For didactic purposes, we explicitly decorate the nodes of the tree with the corresponding semantic representations (although in general they are sets of lambda terms instead of a single one) and with the set of free variables occurring in the corresponding subtree.

(19) s: John likes himself



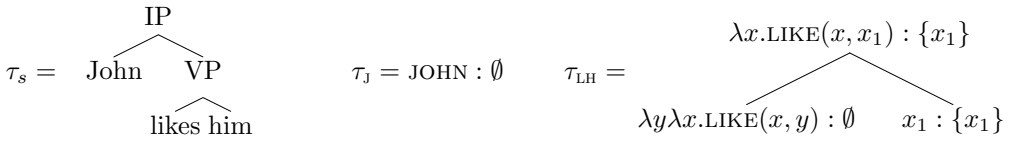
Trees τ_{J} and τ_{LH} are the results of first stages of bottom-up computations, where no c-command relation holds between type e nodes of the tree. In order to get the full semantic representation of the sentence we have to compute $\tau_{\text{JLH}} = \tau_{\text{J}} \cdot \tau_{\text{LH}}$. Tree τ_{J} if of type e, so we are in the situation described in the previous section. Since $\text{FV}(\tau_{\text{LH}}) = \{x_1^*\}$ and $local(\tau_{\text{JLH}}, \text{JOHN}, x_1^*) = TRUE$ this is an example that must

be dealt with by case 1a: therefore QR is triggered over JOHN, leaving x_1^* variable as its trace.

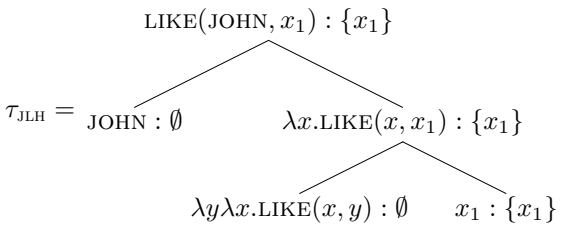


Tree τ_{JLH} is the result of the new definition of functional application between trees τ_{J} and τ_{BH}. The fact that one of the two arguments is a DP triggers the operation described in the previous section, which corresponds to a Quantifier Raising for the DP *John* in the phrase-marker. The resulting logical form $[\lambda x_1^*.\text{LIKE}(x_1^*, x_1^*)](\text{JOHN}) = \text{LIKE}(\text{JOHN}, \text{JOHN})$ is the only acceptable for s.

(20) s: John likes him

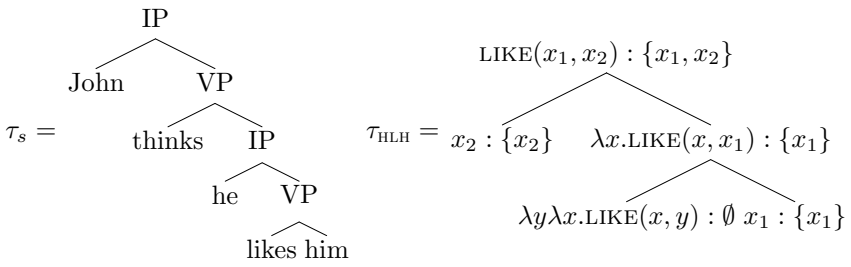


In this case, in computing $\tau_{\text{JLH}} = \tau_{\text{J}} \cdot \tau_{\text{LH}}$ the algorithm detects the situation described at point 2a: $local(\tau_{\text{JLH}}, \text{JOHN}, x_1) = TRUE$. According to principle B, no semantic binding must be generated between *John* and *him*, so QR must be blocked in the present configuration. The parse tree does not get rearranged and the semantic interpretation process goes on the standard way:

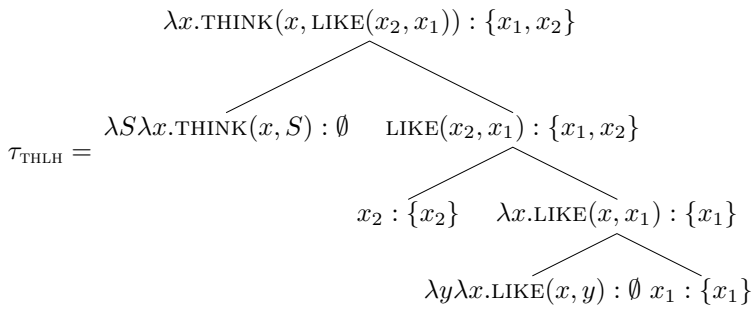


The standard logical form $\text{LIKE}(\text{JOHN}, x_1)$ is then computed for the sentence.

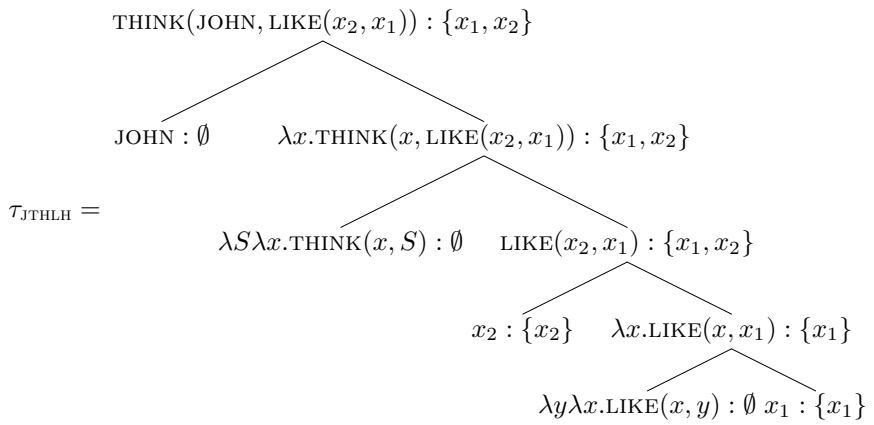
(21) s: John thinks that he likes him.



As an application of principle B, the interpretation algorithm must guarantee that no binding occurs between *he* and *him*. Besides this constraint, nothing is stated so the interpretation algorithm generates semantic binding whenever possible: namely, between *John* and *he*, and between *John* and *him*. So we expect as the output of the interpretation process three readings: one in which *John* binds *he*, one in which *John* binds *him*, and one in which *John* binds neither of them. The computation of parse tree τ_{HLH} for the fragment *he likes him* requires a functional application step between the type *e* DP *he* and VP *likes him*, which falls under the 2a case, and as such it does not entail any binding between *he* and *him*. Without loss of generality, we forget about intensional subtleties introduced by the opaque context created by verb *think* and we assume that the semantics for the fragment *thinks that he likes him* is given by τ_{THLH}:

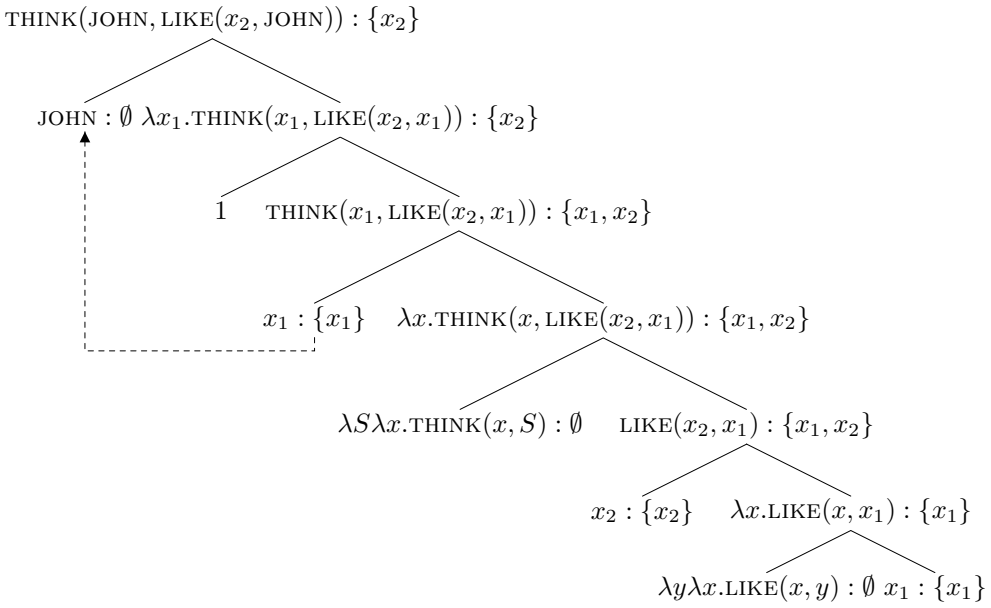


The semantic representation for *s* is given by $\tau_{\text{JTHLH}} = \tau_{\text{J}} \cdot \tau_{\text{THLH}}$, where the type *e* DP *John* gets combined by functional application with the VP *thinks that he likes him*. In this case the DP *John* c-commands both elements of the set $\{x_1, x_2\}$, with $local(\tau_{\text{JTHLH}}, \text{JOHN}, x_2) = FALSE$ and $local(\tau_{\text{JTHLH}}, \text{JOHN}, x_1) = FALSE$. According to point 2b, for each of the free pronouns that occur in τ_{THLH} an additional reading must be created in which *John* binds it, in addition to the default reading in which no semantic binding occurs at all.



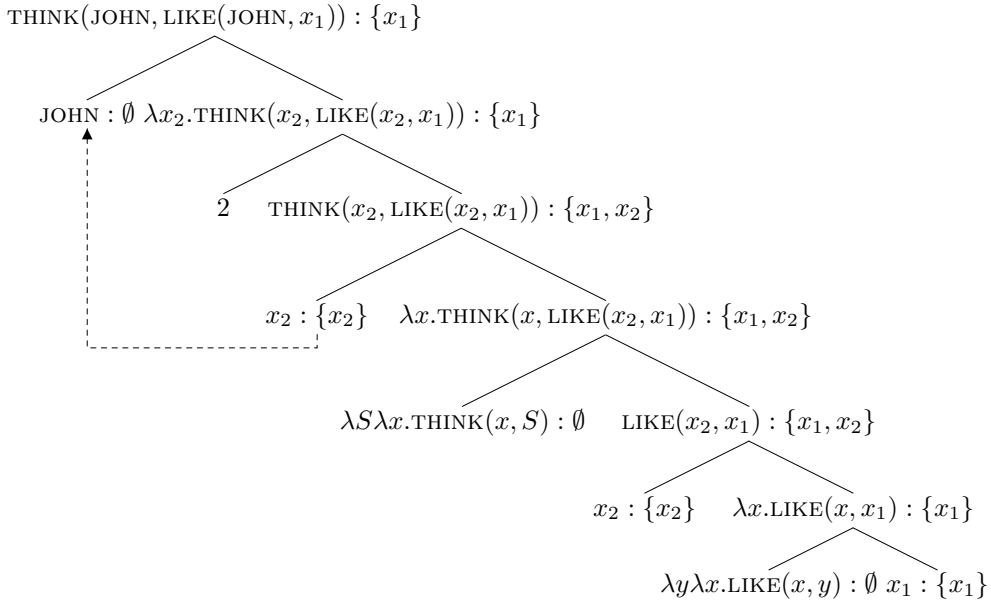
The semantics for τ_{JTHLH} is compatible with the situation in which *John*, *he* and *him* can refer (although not necessarily) to three distinct individuals.

The algorithm generates an additional reading to account for optional binding between DP *John* and pronoun *him*, that is between JOHN and x_1. Since $local(\tau_{\text{JTHLH}}, \text{JOHN}, x_1) = FALSE$, we are in a situation which falls under point 2b. This triggers QR for *John*, leaving x_1 as a trace, resulting in τ'_{JTHLH}:



Parse tree τ'_{JTHLH} corresponds to a reading in which *John* and *him* are the same individual, and that there is a (possibly) distinct singular male individual by whom John thinks to be liked.

The last reading, in which John thinks to like someone else (male, singular), is generated in an analogous way, always according to point 2b. The only thing that changes is that QR leaves a variable which corresponds to *he* instead of *him*, that is the trace left is x_2 instead of x_1, and the abstraction is made on that variable.

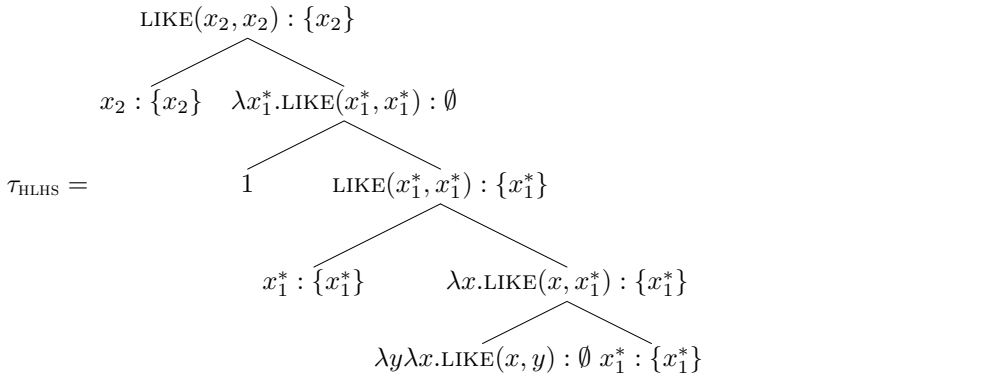


As it is apparent from the example, in no case we can end up with a bound variable reading between the two pronouns *he* and *him*, as expected.

One last variation is worth considering for this sentence schema, that is when the sentence is *John thinks that he likes himself*. In this case we want to end up with only two readings: one in which John thinks that he has the property of liking himself, and another in which he believes that a (possibly distinct from him) male singular individual has that same property. We easily verify that our algorithm yields both desired readings, and only them.

(22) *s*: John thinks he likes himself

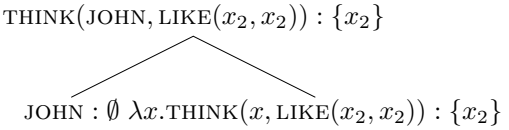
The computation generates in a bottom-up fashion the parse tree τ_{HLHS} for the fragment *he likes himself*. When composing by functional application *he* and *likes himself* it comes across a 1a case which triggers QR for *he*. The result is τ_{HLHS}:



This is the only possible reading for the fragment *he likes himself*. So the semantics for the VP *thinks that he likes himself* is $\tau_{\text{THLHS}} = \lambda x.\text{THINK}(x, \text{LIKE}(x_2, x_2))$,

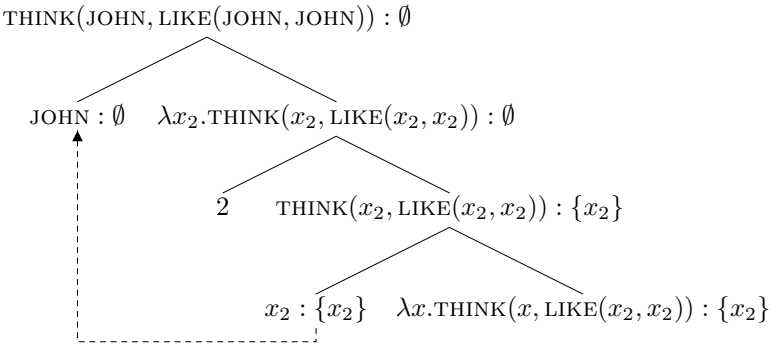
with $FV(\tau_{\text{THLHS}}) = \{x_2\}$. The resulting semantic representation for the full sentence comes from the functional application of DP *John* to VP *thinks that he likes himself*. The algorithm detects a c-command relation between JOHN and the only free variable in τ_{THLHS}, that is x_2. We have that $local(\tau_{\text{JTHLHS}}, \text{JOHN}, x_2) = FALSE$, therefore we are in case 2b. This means that on one hand the algorithms generate a logical form without semantic binding, which corresponds to the simple functional application without QR, and that generates the logical form $\text{THINK}(\text{JOHN}, \text{LIKE}(x_2, x_2))$ corresponding to τ_{JTHLHS}:

$\tau_{\text{JTHLHS}} =$



on the other hand the algorithm generates an additional reading in which semantic binding occurs between JOHN and x_2, yielded by the following tree issued from QR:

$\tau'_{\text{JTHLHS}} =$



4.4.7 Observations

In switching from the coreferential interpretation of the principles of Binding Theory to Reinhart's bound variable approach we gained some important insights, both conceptual and computational.

The algorithm sketched in section 4.3 aims at encoding in the final semantic representation for the sentence a set of constraints on the *denotations* of the entities involved. Under many respects, it is a plain translation of the stipulations of classical Binding Theory into an inductive computational framework. However, the technical implementation showed some important shortcomings from an algorithmic point of view. Such limits and complications were taken as clues that the conceptual approach behind the algorithm was somewhat flawed.

By contrast, the semantic interpretation procedure just presented is driven by the need to generate all possible readings of a sentence, in particular to recover and explicit *bound-variable readings* between pairs of DPs that happen to be in a c-command relation. We conjectured that the way by which the interpretation algorithm generates such readings is a specific application of Quantifier Raising. QR is in general triggered every time the algorithm detects an unbound pronoun or trace c-commanded by another type e DP. We may say that its goal is to equate variables and bind them by the same lambda operator. QR is the operation by

which this effect is achieved. Such a general computational principle, together with a computationally reasonable assumption of *cyclicity* naturally accounts for dropping principle C.

The correspondence between pronouns and variables in lambda calculus has proved to be a fruitful one. Not only free pronouns in a natural language sentence seem to behave like free variables in a formula in a logical language; but the behavior of semantically bound pronouns in the sense of Reinhart's approach to Binding Theory closely reminds of the behavior of bound variables in the sense of standard logic. This is not trivial if we consider that the two notions of binding have evolved independently within two domains that didn't show any particular connection up to the work of Montague and the birth of modern formal semantics for natural language.

What is most interesting to us is that this correspondence seems much more natural if we adopt Reinhart's approach to Binding Theory. By giving up to a vision in which Binding Principles rule the relationship between linguistic entities and real world denotations to embrace one in which they rule an internal, purely formal notion as semantic binding, we obtain a much more elegant computational framework. The other way round, such kind of computational elegance justifies on computational ground the idea that BT principles deal with the purely formal property of semantic binding instead of mutual coreference relations between DPs occurring in a sentence.

It must be said however that we apparently give up some information that the coreferential approach seems to give us: at the present stage of implementation, in this approach we can't account anymore for the fact that, except for quite exceptional contexts, when a human speaker utters the sentence *John blames him*, he wants to convey the idea that John blames some male individual distinct from himself. A computationally acceptable solution to this problem will be provided in section 4.5.

Just like good software engineering is more about correctly devising smart architectures for accomplishing a task, than about accomplishing the task in itself, so we believe that Reinhart's approach, and our computational implementation, represent a progress in devising a computational model of an optimal division of labor between different levels, or layers, of human language faculty.

In particular we believe that by pushing even further the connection between pronouns and variables, and that between binding in syntax with binding in logic, we can gain even better understanding and better algorithms. One notion that stubbornly resists our inductive approach is the notion of binding domain. Linguistic literature about Binding Theory still lacks a satisfactory definition of local domain for a pronoun which both has empirical coverage and nice inductive definitions. If the connection with logics has been a good guide until now in giving us a better algorithm and provide further evidence that Reinhart's approach is a better one with respect to the classical, coreferential one, it makes sense to ask ourselves what is the notion that correspond to local domain in logic. The answer, we believe, leads quite naturally to an implementation of Binding principles which integrates some of the key intuitions of Chomsky 1981, Reinhart 1983, and Reinhart and Reuland 1993, which is the object of next section.

4.5 An Integrated Approach to Binding Theory

In this section we depart from the somewhat mechanical task of faithfully implementing different linguistic interpretations to Binding Theory, to move towards integrating in a unified computational framework features drawn from each of them, which are combined to overcome each other's shortcomings. The result is an original integrated algorithmic treatment of Binding Theory which is both computationally viable and linguistically well-grounded, based as it is on the major insights of last 30 years of linguistic enquiry on this matter.

Our computational approach is particularly indebted with two of the most influential interpretations of Binding Theory. We owe to Reinhart's approach of [50] the divorce of binding principles from coreferentiality issues, and the idea that the latter should be dealt with at a pragmatic/discourse theory level. From a technical point of view, we inherit from our implementation of section 4.4 the idea that the semantic interpretation algorithm must be modified to generate whenever it's possible bound-variable readings between pairs of type e DPs occurring in c-command positions in a phrase-marker. The main difference with the previous approach lies in the fact that the generation of such readings is "blind" to binding domain boundaries.

In the previous sections we sketched two algorithmic approaches to integrate in the semantic interpretation process the principles of Binding Theory. They were directly inspired in one case (section 4.3) by the classic coreferential approach of Chomsky and Lasnik, and in the other (section 4.4) by Reinhart's "bound-variable" interpretation. What they both lack is a neat computational treatment of the notion of binding (or local) domain for a pronoun: in section 4.2 we made the (strong) assumption that we can always tell by means of a binary predicate *local* whether two pronouns occurring in the same parse tree belong to the same local domain or not. We didn't delve deeper into the technical details of the implementation of such a decision procedure to concentrate on the formal machinery to generate the final semantic representations. The problem is that we still lack a satisfactory characterization of the notion of local domain which guarantees both empirical coverage and a simple inductive characterization. Reinhart and Reuland of [53] and [54] allow us to (partially) overcome this basic flaw by replacing the syntax-based, computationally unappealing notion of local domain with the semantics-based criterion of *co-argumentality* (i.e. one in which the fact that two DPs are the arguments of the same predicate plays a decisive role). By integrating their main insights on structural constraints on the reflexive nature of predicates, we are able to filter the unacceptable readings that might be generated during the previous phase without resorting to any definition of binding domain.

Another important issue, raised by the implementation of Reinhart's approach in section 4.4, was left unaddressed. Reinhart's "economy principle" encoded by Rule I (see section 4.4.2) translates algorithmically into an instantiation of an NP-complete problem of complexity theory. With the help of the insights that come from Reinhart and Reuland's approach we are able to address and solve this problem in computationally tractable terms.

4.5.1 Reinhart and Reuland's "Reflexivity" approach

In their 1993 article "Reflexivity" [54], Reinhart and Reuland operate a major shift in the interpretation of Binding Theory principles. They advocate a radically more semantic approach, grounded on the notions of *reflexive predicate* and *reflexive-marker*. Their proposal amounts to a return to a more traditional view of anaphora phenomena, according to which conditions A and B govern the well-formedness of the interpretation of *reflexive predicates* in natural language, instead of coreferentiality or semantic binding between DPs. We shortly recall here the basic definitions and conditions A and B, presented in greater detail in section 3.4:

Reflexive predicates. A predicate P is *reflexive* if and only if two of its arguments are identical, as in $\lambda x.P(\ldots, x, \ldots, x, \ldots)$.
Reflexive-markedness. A predicate P is *reflexive-marked* if and only if either P is lexically reflexive or one of P's arguments is a SELF-anaphor.

(23) a. John$_i$ admires himself$_i$
b. *John$_i$ admires him$_i$

The predicate *admire* in (23-a) is reflexive because two of its arguments are necessarily identical, while clearly this is not the case for (23-b). The semantic structure of the predicate in (23-a) is indeed $\lambda x.\textsc{admire}(x, x)$, while for (23-b) it's $\lambda x \lambda y.\textsc{admire}(y, x)$. In the former the *variables* corresponding to subject and object arguments are identical, in the latter they are not. Note that nothing prevents the *values* that will saturate variables at the interpretation phase from being identical, like in $[[\lambda x \lambda y.\textsc{admire}(y, x)](\textsc{john})](\textsc{john}) = \textsc{admire}(\textsc{john}, \textsc{john})$.

On the basis of these two notions, Reinhart and Reuland redefine traditional Binding Theory principles as the two following conditions:

Condition A. A reflexive-marked predicate is reflexive.
Condition B. A reflexive predicate is reflexive-marked.

The basic idea is that a universal property of natural language is that reflexivity of a predicate must be explicitly licensed at the morphosyntactic level. A predicate can be reflexive only if it is linguistically marked as such and the only two available ways to mark reflexivity are marking the predicate's head (which is the case for intrinsically reflexive predicates like, for example, English verb *wash* in its intransitive use), or marking one of the arguments by using a reflexive pronoun. In (23-a) the predicate is reflexive and the presence of a reflexive pronoun as one of its arguments marks it as reflexive. Both condition A and condition B are thus met. In (23-b), on the contrary, the predicate is reflexive (coindexing here encodes the fact that its two arguments are identical) but nothing in morphology or syntax marks it as reflexive-marked. The violation of condition B therefore makes the semantic interpretation fail.

For our computational purposes, what is particularly interesting is that this new approach makes no use of configurational relations like c-command and binding domain. Binding conditions are expressed strictly as conditions on reflexive predicates and their reflexive-markedness, regardless of their internal structure. This remarkable change in perspective eliminates one of the major drawbacks of

our previous approaches, that is the algorithmic implementation of the notion of local domain for a Determiner Phrase.

Condition A in this new formulation establishes a correspondence that goes from the syntactic level to the semantic one: any predicate marked as reflexive at the syntactic level must translate into a semantically reflexive one, i.e. one in which at least two of its arguments are identical and bound by the same lambda operator.

Condition B works the other way round, from semantic to syntactic level: any semantically reflexive predicate must derive from a reflexively marked syntactic predicate. Such kind of constraint will be implemented twofold. On the one hand, as in section 4.4.5, every occurrence of a pronoun in a sentence will be mapped into a fresh new variable. This ensures that the predicates that are used during the semantic computation of an arbitrary sentence are going to be considered by default non-reflexive (since there are no two occurrences of the same variable) and can be made reflexive only by an explicit mechanism triggered by condition A (predicate marked as reflexive). On the other hand, we establish a well-formedness criterion on the logical forms yielded by the semantic computation that prevents our system from generating semantic representations in which predicates which are not reflexive-marked in the syntax are used reflexively. Conditions A and B together establish a bijection between reflexive predicates at the semantic level and reflexive-marked predicates at the syntactic level.

We basically adhere to the idea of Baauw and Delfitto [2] to consider Principle B as an interface filter that prohibits arity reduction in syntax. The idea is that a *relation* (i.e. a two-place predicate) cannot be reduced to a *property* (i.e. a one-place predicate) as a result of the operations performed within the computational system underlying human language, or in the course of the interpretation process, unless it is already marked as a property in the lexicon. A sentence like *John likes him* cannot be interpreted as $[\lambda x.\textsc{like}(x, x)](\textsc{john})$ since this interpretation entails that the *relation* $[\![\textit{likes}]\!] = \lambda x \lambda y.\textsc{like}(y, x)$ would have been converted into a *property* without explicit license at the morphosyntactic level by means of a reflexive marker.

4.5.2 Additional apparatus

In order to implement a computational treatment of Binding Theory which integrates the basic insights of Reinhart and Reuland's approach, we need to slightly enrich our logical language with some additional formal apparatus and suitable interpretation constraints.

We distinguish in our logical language three kinds of variables: regular variables x_i, (reflexive-)marked variables x^*_j, and logophoric variables x^σ_k. Non-reflexive pronouns and reflexive pronouns can be identified as such on the basis of morphosyntactic features: therefore, non-reflexive pronouns are immediately translated into regular variables and reflexive pronouns into marked variables. The mark '*' can be considered as a particular feature of the variable among others (like gender and number) that we highlight because it plays a specific role in our semantic computations. Logophors are seen in our approach as reflexive pronouns which fail to mark as reflexive the predicate in which they occur (we'll detail in section 4.5.3 under which conditions this happens). Logophors are morphologically

indistinguishable from reflexives, so at first they will be translated into our logical language as reflexive-marked variables. They are identified as logophors (and thus turned into σ-marked variables) at a later stage of the interpretation procedure.

Every predicate P of our logical object language comes either in a normal or a (reflexive-)*marked* version. Let P be a syntactic n-ary predicate and P its logical form translation (i.e. such that $[\![\mathsf{P}]\!] = \lambda x_1 \ldots \lambda x_n.P(x_1, \ldots, x_n)$). If P is an intrinsically reflexive predicate, or if one of its arguments is a reflexive pronoun, its denotation gets marked as P^*. Reflexive pronouns reflexive-mark any predicate in which they occur as one of the arguments.

Reflexive-marking of a predicate by a reflexive-marked argument.
When a predicate P is applied to a reflexive-marked variable x_i^*, the variable passes its mark to the predicate:

$$[\lambda x.P(x_1, \ldots, x, \ldots, x_n)](x_i^*) = P^*(x_1, \ldots, x_i, \ldots, x_n)$$

The reflexive mark of a predicate is a meta symbol that imposes the following well-formedness constraints on the logical structure of a predicate:

Constraint 1: let P be a syntactic n-ary predicate; if

$$[\![\mathsf{P}]\!] = \lambda x_1 \ldots \lambda x_n.P^*(x_1, \ldots, x_n)$$

then there exists a variable x_i that occurs at least twice in the signature of the predicate; that is there exists an i such that

$$[\![\mathsf{P}]\!] = \lambda x_1 \ldots \lambda x_i \ldots \lambda x_n.P^*(x_1, \ldots, x_i, \ldots, x_i, \ldots, x_n).$$

If no such i exists, the interpretation procedure fails.

Constraint 2: for no i do we have that

$$[\![\mathsf{P}]\!] = \lambda x_1 \ldots \lambda x_i \ldots \lambda x_n.P(x_1, \ldots, x_i, \ldots, x_i, \ldots, x_n)$$

If such an i exists, the interpretation procedure fails.

Constraint 1 translates into our language, enriched with reflexive-marked predicates and variables, Reflexivity's Condition A, according to which every reflexive-marked predicate must be reflexive. If a predicate is reflexive-marked at the morphosyntactic level, its semantic counterpart must be a reflexive predicate; if it's not, the semantic computation fails. We will see in the following sections how this constraint is verified by construction during the interpretation process of a sentence. In short, for every n-ary predicate occurring in a sentence, we generate a corresponding logical form in which two of its arguments are identical and bound by the same lambda operator, thus making the predicate reflexive. For example, as we will see in section 4.5.8, in processing the sentence *John likes himself* our interpretation procedure will generate both the logical form $[\lambda x.\textsc{like}^*(x, x)](\textsc{john})$ and $\textsc{like}^*(\textsc{john}, x)$. The former meets both Constraint 1 and Constraint 2 and is accepted; the latter violates Constraint 1 and is therefore ruled out.

Constraint 2 implements Reflexivity's Condition B: if a predicate is reflexive, it must be reflexive-marked. If we consider the inverse implication, this is equivalent to stating that in an unmarked predicate there are no pairs of identical arguments:

if every reflexive predicate must be reflexive-marked in the syntax, no unmarked predicates can be reflexive. If one is, the logical form must be considered ill-formed and the interpretation procedure fails. Constraint 2 acts as a filter to rule out reflexive logical forms which may be blindly generated out of non reflexive-marked syntactic predicates. As will be made clear in section 4.5.7, for a sentence like *John likes him* our algorithm blindly generates both $[\lambda x.\textsc{like}(x, x)](\textsc{john})$ and $\textsc{like}(\textsc{john}, x)$ as its possible semantic translations. However, the first contains a reflexive predicate which is not reflexive marked: therefore it does not comply with Constraint 2 and it is discarded.

The mark on denotation of predicates is a simple device to keep track of where the semantic predicate comes from (either from a reflexive-marked syntactic predicate or not). A predicate which is not issued from a reflexively-marked syntactic predicate cannot be reflexive, that is, the same variable cannot occur twice among the arguments of a predicate. We are not aware of the existence of more than 3-place predicates either in syntax or semantics: this means that we can make the assumption that the arity of the predicates we will be dealing with is at most 3. Such kind of check can thus be easily performed in constant time.

4.5.3 Logophors

As we have anticipated in section 3.4, not every occurrence of a reflexive pronoun yields reflexive-marking for the predicate in which they occur. When this does not happen, we say that the reflexive pronoun is used as a logophor, and we explicitly mark such a use of reflexive pronouns in our logical language. Logophors can be seen as reflexive pronouns used as a special kind of DPs, which carry their own interpretative features which are ruled by discourse theory or pragmatic criteria.

Our computational treatment of logophors is inspired by Reuland [55]. Informally stated, if in interpreting a reflexive pronoun α the regular semantic strategy is blocked, the interpretation of α by directly accessing the knowledge base becomes available. This is what happens (i) when an anaphor is embedded in a larger argument and does not occur as a direct argument of a predicate and (ii) when α occurs as the only argument of a unary predicate. In Reuland's words, in both of these configurations "a chain between DP and an anaphor in the position of α cannot be formed in principle. It follows that there is no way to encode a dependency between α and a possible antecedent in the syntax. This opens the way for a pronominal interpretation of α".

We say that in such a configuration a reflexive-marked pronoun fails to reflexive-mark the predicate it occurs within and in our system its logical form is turned into a σ-variable. Such variables implement the logophoric use of reflexive pronouns in our system. Actually, they're not real variables since they cannot be bound: as logophoric, they come with interpretation constraints on their own that depend on the context of utterance of the sentence.

Logophoric use of reflexives.

When a reflexive-marked variable gets semantically combined in any other contexts than as the argument of a n-ary predicate, with $n \geq 2$, it's turned into a σ-variable: $\beta = [\![\alpha]\!] \cdot x_i^* = [\![\alpha]\!] \cdot x_i^\sigma$

4.5.4 On Rule I

The system we presented in section 4.4 was faced with a linguistic property that it couldn't account for in computational terms, namely that a sentence like *John likes him*, when uttered in a normal context (and not a "pathological" one like Oscar sentences), means that John likes some singular male individual *different* from himself. By choosing to adhere to Reinhart's view of Binding Principles as only governing semantic binding instead of imposing denotational constraints on DPs, we implemented an inductive procedure that generates the logical form $\textsc{like}(\textsc{john}, x)$ as the only semantic representation for the sentence. However, we were left in the dark about the efficient implementation of a computational criterion that favors assignments for x such that x is *not* mapped into JOHN. This is the kind of evidence that Reinhart in [50] accounts for by introducing Rule I. In section 4.4.2 we have seen why her solution is not viable from a computational point of view, and therefore this fundamental issue was put on hold in our implementation of Reinhart's approach in section 4.4. The computational system described in section 4.3 explicitly forbade coreference within such contexts, but at the price of a cumbersome additional apparatus for the semantic representations and several computational redundancies.

Now the constraints on the well-formedness of predicates introduced in section 4.5.2 enable us to propose a solution which is both computationally feasible and linguistically motivated, with some far-reaching consequences that we are going to detail in the following sections.

Let's consider the basic example quoted at the beginning of this section, together with two possible logical forms:

(24) a. John likes him.
b. $[\lambda x.\textsc{like}(x, x)](\textsc{john})$
c. $\textsc{like}(\textsc{john}, x)$

In example (39) we detail how the basic version of our algorithm generates "blindly" both (24-b) and (24-c) as possible semantic representations for (24-a). We stipulate that the two readings are generated in a specific order, namely, that the bound-variable reading is generated first. This can be justified on the basis of "economy" principles: semantic binding is an internal, purely formal device of the computational system of human language, and as such one which is likely to be less computationally demanding than the "coreferential module" which may be responsible for the free variable reading (this is also the position endorsed by Reuland [55]). In the case of sentence (24-a), this means that (24-b) is generated first by the algorithm. Since predicate LIKE does not bear the reflexive mark, and two of its arguments are identical variables, this logical form is deemed unacceptable on the basis of principle B (i.e. a violation of Constraint 2 of section 4.5.2). We push this unacceptability judgement a little further. By a single step of β-reduction we get:

(25) $[\lambda x.\textsc{like}(x, x)](\textsc{john}) \stackrel{\beta}{\Longrightarrow} \textsc{like}(\textsc{john}, \textsc{john})$

Since the left-hand side of (25) has been found unacceptable as a semantic reading for (24-a), it makes perfectly sense to infer the unacceptability of the right-hand

side as well, which is obtained from (24-b) by means of an elementary operation of lambda calculus (β-reduction in this case). We state the following rule:

Unacceptability of β-reduced logical forms.

Let s be a (fragment of a) sentence, $\alpha = [\![s]\!]$, and β the denotation of a DP occurring in s. Let α' the result of applying QR over β in α, that is $\alpha' = [\lambda x.\alpha(x)](\beta)$. If $[\lambda x.\alpha(x)](\beta)$ is unacceptable as a logical form for s, $\alpha(\beta)$ must be considered unacceptable as well.

The intuitive idea is that if, by applying Quantifier Raising over a DP occurring in a given sentence, we end up generating a logical form α' which does not comply with Constraints 1 or 2 from section 4.5.2, such unacceptability judgement must be extended to the logical form obtained from α' via a single step of β-reduction. Simply-typed lambda calculus (on which our elementary semantics is grounded) being strongly normalizable (see [23]), we know that the β-reduction step always terminate.

Furthermore, we assume that any unacceptable reading α' generated during the computation is stored in a working memory $\mathcal{D}$ that can be accessed by the cognitive system at any time. This will be used to constrain possible assignment functions applied to free variables in the correct logical form γ for the sentence.

Let's get back to (24-a). Since the interpretation process has detected a violation of Constraint 2 for (24-b), the algorithm goes on with the alternative derivation, which leads to compute (24-c) as the possibly correct logical form. In (24-c) predicate LIKE is neither reflexive-marked nor reflexive, therefore no violation of Reflexivity Condition A or B occurs and this logical form is accepted.

We're now faced with the problem of explaining why any assignment g for (24-c) such that $g : x \mapsto \textsc{john}$ must be considered as undesirable for (24-a), and how to implement that in computational terms. The answer, we believe, lies in the fact that $\textsc{like}(\textsc{john}, \textsc{john})$ is a logical form which has *already* been excluded as a possible reading for (24-a) because it is the result of β-reduction applied to an incorrect logical form, as shown in (25), and as such it is already stored in the set $\mathcal{D}$ of "forbidden logical forms". Any assignment g such that $x \mapsto \textsc{john}$ makes the open formula (24-c) "precipitate" into $\textsc{like}(\textsc{john}, \textsc{john})$, which belongs to $\mathcal{D} = \{[\lambda x.\textsc{like}(x, x)](\textsc{john}), \textsc{like}(\textsc{john}, \textsc{john})\}$ and therefore it is unacceptable. We formalize our informal observations on the constraints for the assignment function as follows:

Unacceptable assignments rule.

Let s be an expression with the set $\mathcal{D}_s$ of forbidden logical forms associated, and α a logical form such that $[\![s]\!] = \alpha$. Then any assignment function g such that $\alpha^g \in \mathcal{D}_s^g$ is not acceptable[4].

Our approach is based on reasonable cognitive assumptions: the logical form involving semantic binding must be generated first, and if a logical form is recognized incorrect any other one derived from it by a single step of β-reduction is considered incorrect as well. This accounts for the restrictions on the assignment for free variables: it wouldn't make any linguistic sense to reject on the basis of "reflexivity

[4] Quite intuitively, we indicate with $\mathcal{D}_s^g$ the set of logical forms $\beta_i = \alpha_i^g$ for every $\alpha_i \in \mathcal{D}_s$.

issues" a logical form which is later accepted when an undistinguishable semantic interpretation is achieved via coreference.

To be entirely faithful to the original insight of Tanya Reinhart, we should also forbid any assignment that maps a free variable in the final logical form into the same denotation that saturates a bound variable. Consider for example the following sentence and one of the possible logical forms associated:

(26) a. John thinks that his father hates him
b. $[\lambda x.\textsc{think}(x, \textsc{hate}(\textsc{father}(x), x_1))](\textsc{john})$

In (26-a) *John* and *his father* are seen as semantically bound, while *him* is a free variable, and as such it can be mapped into any DP, *John* included. However, Reinhart's argues that in this configuration *him* shouldn't be allowed to corefer with *John*: in the case the speaker wanted to convey the meaning that John thinks that he not beloved by his father he would have privileged the bound-variable reading for both pronouns. Although technical feasible, we believe that for the purpose of our present treatment of pronouns, making explicit such kind of constraints results in a for the moment unnecessary cumbersome representation and we leave this task for a future implementation.

4.5.5 Principle C effects

The mechanism used to exclude the coreferential reading described in the previous section can be applied in a more general way to derive obviative effects formerly due to Principle C.

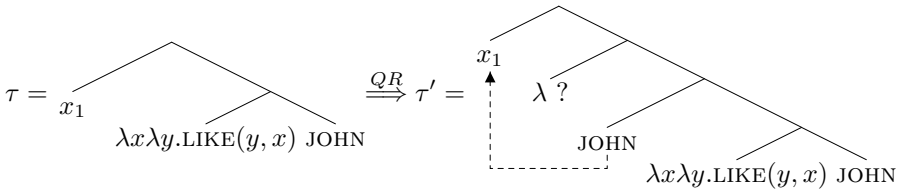
(27) a. He likes John
b. He thinks that Bill hates John

In the framework of traditional Binding Theory the application of principle C prevents *he* from coreferring with *John* in both sentences. In the computational implementation presented in section 4.3.2 we explicitly express forbidden coreference between the pronoun and the full-DP via obviation pairs. Having given up the coreferential framework in section 4.4 to embrace one in which Binding Theory only rules semantic binding, we are left with the problem of accounting for such kind of empirical evidence without incurring in any computationally intractable problem like those faced by Tanya Reinhart's Rule I. The solution that we adopt is an extension of the approach we sketched in the previous section. We formalize what just said in the following principle:

Lexical obviation. In every configuration when QR can apply, but is discarded on purely lexical grounds, the reading that would be obtained in the same configuration if QR could apply, as well as all readings that can be derived from it through a single step of β-reduction, must be considered unacceptable as well.

In both (27-a) and (27-b) DPs *he* and *John* are in a position that makes semantic binding available, except for the fact that their respective position is inverted: using the classical configurational terminology, the pronoun is in a c-commanding

position, while the full-DP is in a c-commanded position. We might say that they are in a Σ^* configuration, somewhat symmetric to the Σ configuration we previously introduced as the one which licenses Quantifier Raising and thus semantic binding, but with the DP and the pronoun occupying inverse positions. Therefore the lexical choice of the elements is incompatible with the application of Quantifier Raising, and thus with semantic binding. In such a configuration the application of the QR operation introduced in section 4.4.5 does not make sense. We recall that such an operation was introduced to generate a bound-variable reading between a c-commanding DP and a c-commanded pronoun. Since the c-commandee in this case is a full-DP, and as such its semantic counterpart is not a variable, lambda abstraction over something which is not a variable does not make any sense and the semantic bound reading is ruled out.



The fact is that the two DPs are in the typical configuration that licenses QR, but the lexical choices made (the full-DP c-commanded by the pronoun instead of the other way round) are incompatible with it and therefore with the possibility of semantic binding between them. This must be taken as a clue that the speaker wants to convey some form of obviation: not only semantic binding between the c-commanding pronoun and the c-commanded DP is forbidden, but also every logical form which is denotationally equivalent to it as well (according to the principle of unacceptability of derived readings that we introduced in the previous section). Therefore, from this particular configuration we can infer that the following semantic representations (involving semantic binding between the pronoun and the DP) for (27-a) and (27-b) respectively are forbidden:

(28) a. $[\lambda x.\text{LIKE}(x, x)](\text{JOHN})$
b. $[\lambda x.\text{THINK}(x, \text{LIKE}(\text{MARY}, x))](\text{JOHN})$

Following the same line of thought adopted in the previous section, also logical forms $\text{LIKE}(\text{JOHN}, \text{JOHN})$ and $\text{THINK}(\text{JOHN}, \text{LIKE}(\text{MARY}, \text{JOHN}))$, derived via a single β-reduction step from (28-a) and (28-b), must be rejected as semantic representations for (27-a) and (27-b) respectively. Using the terminology introduced in the previous section, for (27-a) we have that

$$\mathcal{D}_a = \{[\lambda x.\text{LIKE}(x, x)](\text{JOHN}), \text{LIKE}(\text{JOHN}, \text{JOHN})\}$$

while for (27-b) we have

$$\mathcal{D}_b = \{[\lambda x.\text{THINK}(x, \text{LIKE}(\text{MARY}, x))](\text{JOHN}), \text{THINK}(\text{JOHN}, \text{LIKE}(\text{MARY}, \text{JOHN}))\}$$

Since QR cannot be applied between *he* and *John*, the only logical forms generated for (27-a) and (27-b) are, respectively:

(29) a. $\textsc{like}(x, \textsc{john})$
b. $\textsc{think}(x, \textsc{like}(\textsc{mary}, \textsc{john}))$

that is, semantic readings in which *he* is mapped into a free variable. As such, nothing in the logical language of lambda calculus prevents logical variable x from being assigned the value JOHN. However, the constraints induced by sets $\mathcal{D}$ make such assignments incorrect, or at least mark them as unlikely. Indeed, $[\textsc{like}(x, \textsc{john})]^{x \rightarrow \textsc{john}} = \textsc{like}(\textsc{john}, \textsc{john})$ already belongs to $\mathcal{D}_a$, and the same holds for $[\textsc{think}(x, \textsc{like}(\textsc{mary}, \textsc{john}))]^{x \rightarrow \textsc{john}} = \textsc{think}(\textsc{john}, \textsc{like}(\textsc{mary}, \textsc{john}))$, which belongs to $\mathcal{D}_b$.

The interface condition that excluded semantic representations which were traditionally considered as entailing principle B violations now allows us to forbid, or at least discourage in normal contexts, readings which were traditionally considered as entailing violations of principle C. Note that the Assignment Rule we introduced in section 4.5.4 is perfectly compatible with a non-coreferential perspective which sees Binding Theory as controlling only the property of semantic binding. Denotational constraints only appear at the level of assignment functions, that is pragmatics/discourse theory level. Our computational treatment of this issue accounts quite elegantly for the interaction between these two distinct levels of language faculty, while keeping clearly separated the domains of influence for each of them.

We have to underline how in the case of (27-a) the coreferential reading should also be redundantly excluded on the basis of reflexivity issues: predicate LIKE in (28-a) is reflexive without being reflexive-marked. This inelegant redundancy may be taken as a hint that it should be possible to provide a more unified treatment of cases likes *John likes him* and *he likes John*. However, at the present stage of development it is not yet clear to us how this might be achieved without losing the possibility to account also for cases like (27-b).

4.5.6 On syntactic and semantic predicates

We believe our approach to "Rule I issues" to have some far-reaching consequences, in particular regarding the distinction introduced by Reinhart and Reuland between syntactic and semantic predicates. We will show that our computational treatment of reflexive pronouns together with our criterion about the unacceptability of logical forms derived by incorrect semantic interpretations allow us to get rid of this strongly stipulative assumption which is both linguistically doubtful and computationally disturbing. In our system we are able to subsume and derive it as a natural consequence of our new computational principles, together with a reasonable modification of the rule about the unacceptability of derived logical forms.

We have seen in section 3.4.6 that Reinhart and Reuland are forced by some disturbing empirical evidence to postulate that condition A applies to *syntactic predicates*, while condition B deals with *semantic predicates*. The problem with condition B is exemplified by the need to account for the ungrammaticality of example (32-b) from chapter 3 that we repeat here:

(30) *The queen$_1$ invited both Max and her$_1$ to our party.

The problem is raised by the fact that *Max and her* do not mark reflexively the predicate *invited* and so there is neither violation of Condition B nor violation of Condition A, and yet the sentence is undoubtedly wrong (under the assumption that *her* refers to the queen). The answer given by Reinhart and Reuland to this puzzle is that the semantic translation of (30) is:

(31) the queen $(\lambda x\ (x \text{ invited Max} \wedge x \text{ invited } x))$

and that condition B applies only at this (semantic) level. Indeed, in (31) the second occurrence of *invited* is a reflexive predicate but it's not linguistically marked as reflexive. This violation of condition B accounts for the unacceptability of the sentence.

Reinhart and Reuland's solution is not entirely satisfactory from two points of view. From a theoretical point of view they're forced to introduce an *ad hoc* stipulation about the domain of application of condition B, only to adapt the theory to such recalcitrant examples. From a computational point of view it seems very disturbing to have condition B "wait" to apply until we reach such a level of semantic description, which is not commonly provided by any syntactic analysis of a sentence. The ideas presented in the previous sections provide a natural approach to this problem.

First a word on notation is in order. Consider the sentence:

(32) *John$_1$ likes him$_1$ and Max

We assume that *him and Max* are analyzed as a single plural argument of predicate *like*. It is perfectly natural to force, although in an intuitively unharmful way, the signature of predicate *like* to accept *sets of entities* as arguments[5]. Leaving formal details aside, we give the following interpretation rules (where P is a unary predicate, and Q and R binary predicates):

$$P(\{x, y\}) \stackrel{\delta}{\Longrightarrow} P(x) \wedge P(y)$$
$$Q(\{x, y\}, z) \stackrel{\delta}{\Longrightarrow} Q(x, z) \wedge Q(y, z)$$
$$R(x, \{y, z\}) \stackrel{\delta}{\Longrightarrow} R(x, y) \wedge R(x, z)$$

We indistinctively call the three occurrences of $\stackrel{\delta}{\Longrightarrow}$ the *distributivity rule* for logical formulas.

As described in section 4.5.4, we assume that in interpreting (32) our algorithm first generates the following logical form (which involves bound-variable reading):

(33) $[\lambda x.\textsc{like}(x, \{x, \textsc{max}\})](\textsc{john})$

There's nothing in the sentence that could mark LIKE as reflexive-marked, which is not a reflexive predicate either: thus no violation of Condition A or B occurs at this point. It comes natural to go on with the derivation by applying the distribution rule over the elements of the set thus getting:

[5] With an abuse of notation, when the argument of the predicate is a singleton for ease of reading we drop the braces notation

(34) $[\lambda x.[\textsc{like}(x, x) \wedge \textsc{like}(x, \textsc{max})]](\textsc{john})$

The first occurrence of LIKE is reflexive without being reflexive-marked. This violation of Condition B makes this side of the computation fail. If we apply β-reduction, by an application of the unacceptability principle defined in section 4.5.4, we infer that also $\textsc{like}(\textsc{john}, \textsc{john}) \wedge \textsc{like}(\textsc{john}, \textsc{max})$ is not an acceptable reading for the sentence. We consider that the unacceptability judgement that we just reached for (34) extends "backward" to (33) too. At this point the algorithm goes on with a free variable reading, where no semantic binding occurs between *John* and *him*. The result is:

(35) $\textsc{like}(\textsc{john}, \{x, \textsc{max}\}) \stackrel{\delta}{\Longrightarrow} \textsc{like}(\textsc{john}, x) \wedge \textsc{like}(\textsc{john}, \textsc{max})$

Both sides of (35) meet constraint 1 and 2, therefore they're acceptable as logical forms for (32). If, as it is the case in (32), *him* is supposed to corefer with *John*, i.e. if $x \mapsto \textsc{john}$ in (35), this clashes against the fact that the algorithmic procedure has already deemed logical form $\textsc{like}(\textsc{john}, \textsc{john}) \wedge \textsc{like}(\textsc{john}, \textsc{max})$ as unacceptable for the sentence.

We have just seen that in order to smoothly account for cases like (32), we have to modify the rule about the unacceptability of derived logical forms first given in section 4.5.4 in two (reasonable) ways: (i) we add distributive rule δ to $\beta-$reduction as elementary lambda calculus operations; and (ii) we have to state that unacceptability judgements extends backwards from the derived logical form to the initial one. We state the following rule:

Unacceptability of derived logical forms via δ-rule.

Let α be a logical form for a given sentence s and α' the result of the application of distributive rule δ over α. If α' is not acceptable as a semantic representation for s, then α is not acceptable either.

Otherwise stated, if starting from a logical form α by means of the δ-conversion rule just introduced we end up with a logical form α' which does not comply either with constraint 1 or 2, both α and α' must be deemed unacceptable. This is not a conceptually different approach from Reinhart and Reuland's, but it naturally follows from independent computational principles that already belong to our system, instead of being stipulated through an artificial distinction between syntactic and semantic predicates, which is not necessary in our system.

Furthermore, we can get rid of the complementary stipulation that Condition A applies only to *syntactic* predicates. We recall that in [54] this stipulation is introduced to account for the acceptability of sentences like the following:

(36) a. John$_1$ likes himself$_1$ and Max.

b. $[\lambda x.\textsc{like}(x, \{x, \textsc{max}\})](\textsc{john}) \stackrel{\delta}{\Longrightarrow} [\lambda x.[\textsc{like}(x, x) \wedge \textsc{like}(x, \textsc{max})]](\textsc{john})$

As in the previous example, we assume that *himself and Max* is the composite (and not reflexive-marked) argument of predicate *likes* which therefore does not get marked as reflexive. Sentence (36-a) is translated by Reinhart and Reuland as (36-b). But in the right-hand side of (36-b) we have a reflexive occurrence of LIKE which is not reflexive-marked. So if Condition A applied at the semantic

level, this derivation would be incorrectly ruled out. But if we assume, as Reinhart and Reuland do, that Condition A applies only at the syntactic level, predicate *likes* is neither reflexive-marked nor reflexive, and the semantic interpretation is acceptable. This is the reason which naturally justifies an effect which was achieved by Reinhart and Reuland by the *ad hoc* stipulation that Condition A applies only at syntactic predicates.

Our analysis of (36-a) does not need to invoke syntactic predicates. We know from section 4.5.3 that if a starred variable does not pass its mark to the predicate, it turns into a σ-variable (that is, it is considered to be used as a logophor). Reflexive pronoun *himself* in (36-a), embedded in the complex argument *himself and Max*, is in one of those configurations described in section 4.5.3 which make impossible for it to reflexively mark the predicate. We know that σ-variables are not free variables and as such they cannot get bound: therefore the reading involving semantic binding between *John* and *himself* won't be generated and the only logical form issued by the application of our algorithm is:

(37) $\text{LIKE}(\text{JOHN}, \{x^\sigma, \text{MAX}\}) \stackrel{\delta}{\Longrightarrow} \text{LIKE}(\text{JOHN}, x^\sigma) \wedge \text{LIKE}(\text{JOHN}, \text{MAX})$

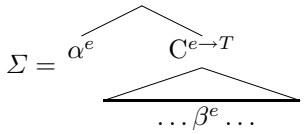
In (37) no violation of Condition A occurs: we consider x^σ to carry independent interpretative features that are ruled by the logophoric use of reflexives and that make it different both from a free and from a bound variable. This is compatible with the view of Binding Theory that we pursue here, that is one which only rule the use of free and bound variables; logophoric use of reflexive pronouns is ruled by an entirely different module, and such kind of pronouns are not subject to usual computational device of BT like Quantifier Raising.

4.5.7 Algorithm's overview

The algorithm is a bottom-up process that inductively computes intermediate representations of the input phrase-marker that can be interpreted by a modified version of the semantic interpretation procedure. For clarity purposes, we present this process as structured into three successive steps: (i) restructuring of the phrase-marker due to optional applications of QR, which in general yields a set of derived phrase-markers; (ii) semantic interpretation for each of them, which results in a set of logical forms; (iii) verification of reflexivity constraints for each of the logical forms computed during phase (ii). However, as it will be made clear later, nothing prevents these three steps to be performed in parallel over the set of phrase-markers possibly generated, during a single bottom-up traversal of the original phrase-marker.

RESTRUCTURING. Step (i) operates on the phrase-marker τ_s issued from a standard generative syntactic analysis for a sentence s. It consists of a bottom-up traversal of the phrase-marker which parallels the computation of standard semantics according to the operations described in chapter 2. With respect to them, additional operations are performed when a specific "merge" configuration is detected, which result in one or more additional phrase-markers issued from the original one. We inherit from the algorithm presented in section 4.4 the goal of the restructuring procedure: to generate semantic binding between Determiner Phrases whenever

this is possible. New semantic binding configurations get established whenever the algorithm detects what we called a $\Sigma-$configuration: a type e DP α which gets merged by means of functional application to a type $e \rightarrow T$ constituent C which contains other type e DPs:



This specific configuration triggers the application of QR between α and all the subsets of the set of DPs β occurring in the subtree C. This results in as many new phrase-markers as the parts of the set of DPs β in C, each of which involving semantic binding between α and β.

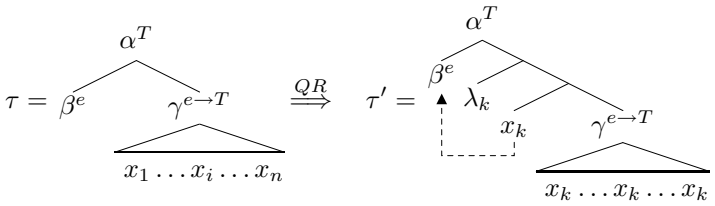
Semantic interpretation. Step (ii) performs standard semantic computations on the phrase-markers issued from step (i), with the addition of a specific rule to characterize the logophoric use of reflexive pronouns. The result is a set of logical forms, one for each of the phrase-markers issued from step (i).

Reflexivity check. Step (iii) consists of a check on the well-formedness of the semantic representations issued from step (ii). In section 4.5.2 we introduced two constraints that logical forms enriched with reflexive-marks must fulfill in order for them to be well-formed. The mechanical application of QR in step (i), blind to former local domains, makes predicates reflexive in the sense of Reinhart and Reuland's Reflexivity. This final check on the correspondence between reflexivity of logical forms and reflexive-markedness of corresponding syntactic forms filters out readings that might have been generated during step (i) and which do not comply with Reflexivity's Conditions A and B (Constraints 1 and 2 in our system).

Step 1: restructuring the phrase-marker

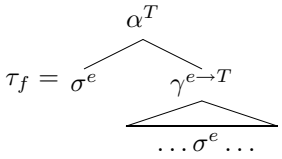
The input to this phase is the phrase-marker τ_s issued from a generative analysis for a given sentence s. We map every pronoun occurring in the phrase-marker into a fresh new variable $x_1, x_2, \ldots x_n$. The output of the restructuring phase is a set $\mathcal{T}$ of phrase-markers τ'_s which are inductively computed as follows for every node α in τ_s:

1. if α is a leaf (i.e. a non-branching node), $\tau'_s = \alpha$;
2. if α is a branching node, and β and γ are α's daughters, then:
   a) if β is a type e DP constituent, for each subset $S \in \mathcal{P}(dps(\gamma))$ (i.e. for each subset of the set of DPs occurring in γ) the algorithm performs the following operations:
      i. every element $x_1 \ldots x_n$ in S which is a free variable with compatible agreement features with β is replaced with a fresh new variable x_k, and a new phrase-marker τ' is created in which β has been Quantifier Raised, leaving a copy of x_k in its trace position as follows:



Phrase-marker τ' is thus added to the set $\mathcal{T}$ of restructured phrase-markers.

ii. for every element σ in S which is not a variable and which has compatible agreement features with β, variable, $\tau' = \alpha$ the semantic interpretation of the following tree:



is added to the set of forbidden readings for the sentence.

b) otherwise $\tau' = \alpha$ (the phrase-marker is copied)

Intuitively, this is a bottom-up procedure which performs the usual semantic computations described in chapter 2 over the original phrase-marker, until when it detects a specific (Σ) configuration: a type e DP β which gets combined (i.e. semantically *merged*) via functional application with another constituent γ which contains other type e DPs.

The algorithm inherits from the one presented in section 4.4 the drive to generate bound variable readings whenever possible. Therefore (point 2(a)i) for each subset of unbound pronouns $x_1, \ldots, x_n$ occurring in γ, an additional phrase-marker is generated in which a bound-variable reading between β and $x_1, \ldots, x_n$ is created by means of an application of Quantifier Raising. For any DP σ occurring in γ which is not a variable (i.e. the DP corresponding to β c-commands a full-DP) (point 2(a)ii), an additional one is created to generate a logical form that we want to forbid.

Point (ii) deserves some further explanation. In section 4.5.5 we have discussed how to implement in the present framework the effects of what was formerly known as principle C. The idea is that since in this configuration QR is forbidden on purely *lexical* grounds, this not only means that any logical form which involves semantic binding between them is not correct, but also that the logical form that would result from such semantic binding must be forbidden. Point (ii) describes an immediate way to compute such a forbidden logical form. Indeed, since σ is not a variable, the only possibility to have semantic binding between β and σ would be to raise σ thus resulting in the logical form: $[\lambda[\![\beta]\!].[\![\alpha(\beta, \sigma)]\!]](\sigma)$, which reduces through a single step of β-reduction to $[\![\alpha(\sigma, \sigma)]\!]$. The same result can be achieved computationally by replacing β with σ in the corresponding parse tree, and the result is τ_f.

Two features deserve to be highlighted in the present approach. First, the notion of c-command has disappeared from our algorithm, paralleling the evolution

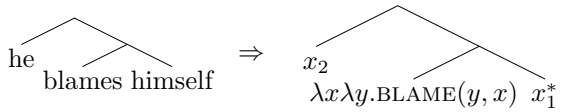
that linguistic interpretations of Binding Theory went through during last decades. What matters to our approach is not an abstract syntactic configuration occurring between two DPs (like c-command is in the classical formulation of binding theory), but the semantic merge occurring between a type e DP constituent and another constituent which contains other type e DPs. Put in this perspective, the notion of c-command is nothing more than an *a posteriori* syntactic description of the result of a much more primitive operation performed at the level of semantic interpretation: the syntactic shadow of a semantic operation. Secondly, this procedure is totally blind to binding domains. As soon as a Σ-configuration is detected, two things may happen: if variable-like expressions are detected in the command-domain (point 2(a)i), the QR applies over each of them, generating as many additional phrase-markers in which semantic binding occurs; when (point 2(a)ii) QR is blocked on purely lexical grounds (a simple switch of c-commander and c-commandee would be enough to restore the conditions for QR to apply), the semantic computation goes on as usual but a "forbidden logical form" is generated, to implement the obviative effect that the speaker achieves by using a configuration that makes QR impossible. In either case, the binding domain of the entities involved plays no role in the computation.

QR is triggered for each DP occurring in the constituent, no matter if it is an anaphor, a pronoun or a full-DP. Needless to say, the application of QR makes sense if and only if the "copied" element is translated semantically into a variable, as we will see during the phase of semantic interpretation, while "raising" of a full-DP will result in an instance of Condition C effects.

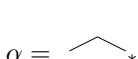
Step 2: interpretation

The semantic interpretation procedure must be sightly modified to implement the ideas sketched in section 4.5.2 on the reflexive marking of predicates and section 4.5.3 on the logophoric use of reflexive pronouns. For every phrase-marker $\tau \in \mathcal{T}$, and for every node α in τ:

1. if α is a leaf:
   a) if α is a reflexive or a non-reflexive pronoun, it is mapped into a fresh new variable; reflexive pronouns bear a reflexive mark;
   b) otherwise, α is mapped into the semantic representation provided by the lexicon;



2. if α is a branching node, in addition to the standard semantic computations we described in chapter 2, we provide special semantic treatment for a configuration that play a special role in our system:



$$\Rightarrow \quad \begin{cases} [\![\alpha]\!] = P^*[x_i/y] & \text{if } [\![\gamma]\!] = \lambda y.P(\ldots, y, \ldots) \\ & \text{and } P \text{ is } n\text{-ary with } n \geq 2 \\ [\![\alpha]\!] = [\![\gamma]\!] \cdot x_i^\sigma & \text{otherwise} \end{cases}$$

Although trivially implemented, point 1a is strategic for the overall interpretation process. By introducing a fresh new variable at each occurrence of a pronoun in a phrase marker, we generate by default the reading in which no semantic binding occurs. Bound variable readings must be explicitly licensed under specific configurations and get possibly filtered out by interface conditions in the next step.

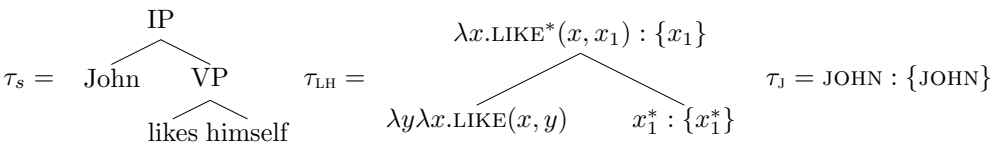
Point 2 implements our treatment of reflexive marking of predicates (see section 4.5.2) and the logophoric use of reflexives (see section 4.5.3). In the first case the reflexive-marked pronoun occurs in an argument position for a predicate: in such a configuration the reflexive mark is passed up to the predicate. In any other configuration the reflexive pronoun occurs embedded into some other kind of semantic constituent: it is the case, for example, of *herself* in a sentence like *the queen invited [Mary and herself] to the party*, or when the reflexive is the argument of a unary predicate. We assume that in such kind of configuration it basically loses its reflexive-marking power and gets marked as a reflexive pronoun used logophorically (and thus it gets semantically translated into what we called σ-variables).

Step 3: interface conditions on reflexivity

The last phase of our interpretation procedure can be thought of as an "interface conditions check" that filters out unacceptable logical forms that were generated during step 1 and 2. The application of QR over c-commanding DPs in a parse tree generates semantic binding between DPs which are in a c-command relation. From Step 1 we know that the restructuring phase applies QR blindly, independently on the binding domain of the DP it applies to. Semantic binding occasionally entails reflexive use for a predicate: if the corresponding predicate is reflexive-marked, the logical form is acceptable, otherwise it gets rejected. Note that QR is basically blind to the type of DPs it applies to: both reflexive and non-reflexive pronouns can be copied in the trace position left behind by a DP which has been QR'd. The only notable exception is represented by full-DPs. Since a lambda-operator cannot bind a full-DP, QR does not apply in such a configuration. We already know that during step 1 a "forbidden logical form" has been computed for this configuration, which will be used to implement condition C effects in constraining the possible assignments to free variables occurring in the final logical form.

4.5.8 Basic examples

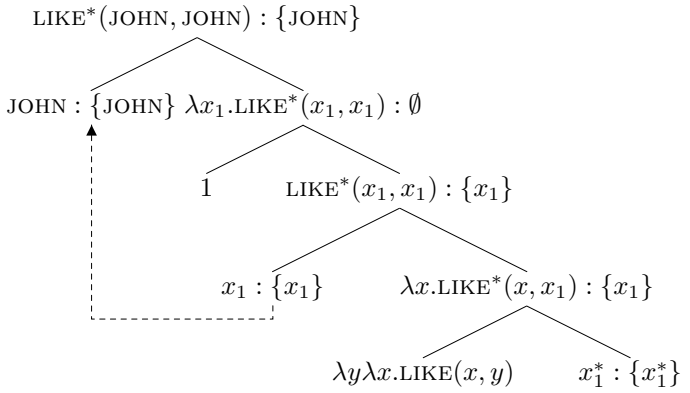
(38) s: John likes himself



In τ_{LH} the reflexive *himself* marks as reflexive predicate *likes*, and this is translated at this level of representation in the mark being passed from the variable x_1^* to the predicate LIKE when *likes* is combined by functional application to *himself*.

When the algorithm performs semantic merge between τ_{J} and τ_{LH}, it detects a $\Sigma-$configuration: type e DP *John* merges with a constituent which contains a type e DP (*himself*) with compatible agreement features. We're in the situation described at point 2(a)i of the previous section. Accordingly, the algorithm performs QR over *John*, copying in the position left by it the semantic representation of *himself*, thus yielding:

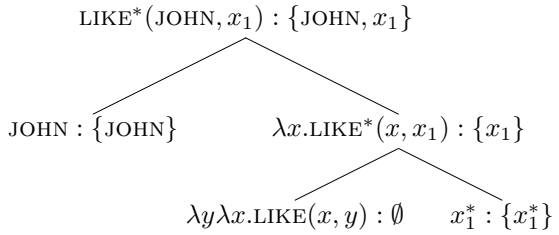
$\tau_{\text{JLH}} =$



When this tree is interpreted at the semantic representation level, it normally yields the logical form $[\lambda x_1.\text{LIKE}^*(x_1, x_1)](\text{JOHN}) = \text{LIKE}^*(\text{JOHN}, \text{JOHN})$. The starred predicate occurs with two identical arguments, condition 1 is thus satisfied and this semantic representation is accepted.

The additional semantic reading generated is the one we get from standard computations when QR does not apply, that is:

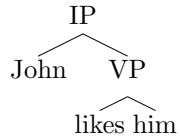
$\tau'_{\text{JLH}} =$



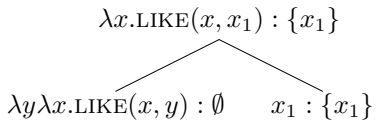
The logical form generated being $\text{LIKE}^*(\text{JOHN}, x_1)$, the interface condition detects a reflexive-marked predicate which is not used reflexively, and thus this logical form is ruled out.

(39) s : John likes him

$\tau_s =$



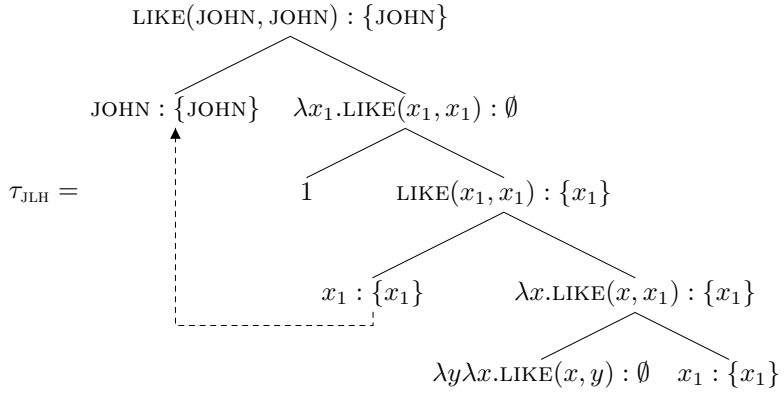
$\tau_{\text{LH}} =$



$\tau_{\text{J}} = \text{JOHN} : \{\text{JOHN}\}$

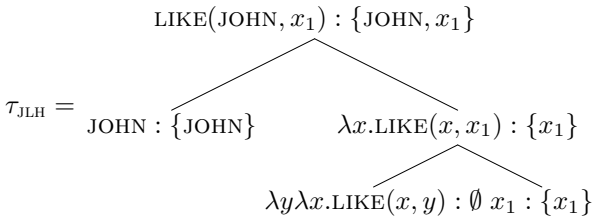
When τ_{J} and τ_{LH} get semantically merged, the algorithm detects a configuration like the one described at point 2(a)i, because of feature agreement between *John*

and *him*. Therefore, QR applies on *John* and copies the denotation of *him* in the trace position left behind, thus yielding the following semantic representation:



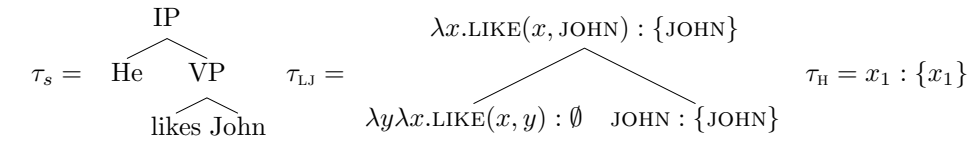
At the interface, an occurrence of a non reflexive-marked predicate with two of its argument identical ($\lambda x_1.\textsc{like}(x_1, x_1)$) is detected in $\tau_{\textsc{jlh}}$. This is a violation of constraint 2 from section 4.5.2, which forbids $[\lambda x_1.\textsc{like}(x_1, x_1)](\textsc{john})$ as logical form corresponding to the sentence *John likes him* as well as $\textsc{like}(\textsc{john}, \textsc{john})$ which is obtained by lambda reduction from it.

Since this interpretation fails, the algorithm goes on computing an alternative semantic representation in which QR does not occur. The result is the following:



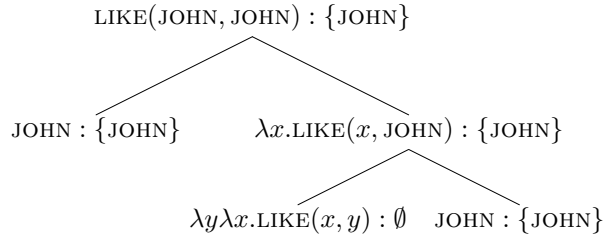
This leads to the semantic representation of *John likes him* as $\textsc{like}(\textsc{john}, x_1)$, where *him* is simply mapped into a free variable. Since this reading is always subordinated to the failure of the bound variable one, we already know at this stage of computation that $\textsc{like}(\textsc{john}, \textsc{john})$ has been recognized as a forbidden reading for this sentence. Thus, any assignment that maps x into JOHN like in $[\textsc{like}(\textsc{john}, x)]^{x \to \textsc{john}}$ clashes against this previous computation. Otherwise stated, the x is free in $\textsc{like}(\textsc{john}, x)$ as long as the assignment does not precipitate the semantic representation into some forbidden logical form previously computed.

(40) s : He likes John



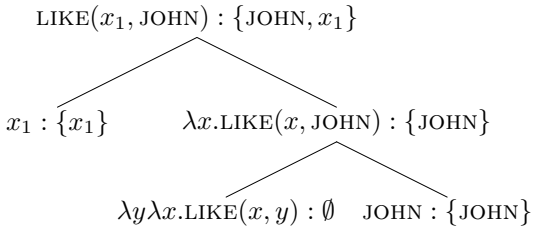
When the algorithm tries to perform the semantic merge between τ_{H} and τ_{LJ} it detects the configuration described at point 2(a)i in previous section: *he* and *John* have compatible agreement features, the former c-commands the latter whose semantic interpretation *is not* a variable but a logical constant. QR of a non-variable semantic element like the logical constant JOHN does not make sense. The two DPs are in a configuration that is similar to the Σ-configuration that in our system licenses Quantifier Raising but their lexical features block this possibility. As we have explained in section 4.5.5, not only semantic binding is forbidden (and thus no QR occurs for this parse tree), but we need to forbid the logical form that would result if such a semantic binding occurred. In this case such a logical form is $[\lambda x.\text{LIKE}(x, x)](\text{JOHN})$, which reduces after a step of β-reduction into $\text{LIKE}(\text{JOHN}, \text{JOHN})$. Such a logical form can be mechanically generated by interpreting the following tree, obtained by copying the denotation of the c-commanded constituent in the position of the c-commanding, as described at point 2(a)ii of the previous section:

$\tau_f =$



The computation for the logical form of the sentence goes on without QR to generate the following parse tree:

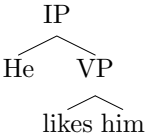
$\tau_{\text{HLJ}} =$



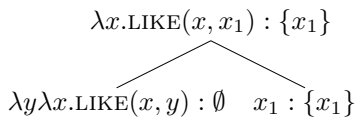
Therefore the correct semantic interpretation for the sentence is $\text{LIKE}(x_1, \text{JOHN})$, for every assignment such that it is not the case that $x \rightarrow \text{JOHN}$. In this case indeed the semantic representation coincides with $[\![\tau_f]\!]$ that we have already computed as an incorrect semantic representation for the sentence.

(41) *s*: he likes him

$\tau_s =$



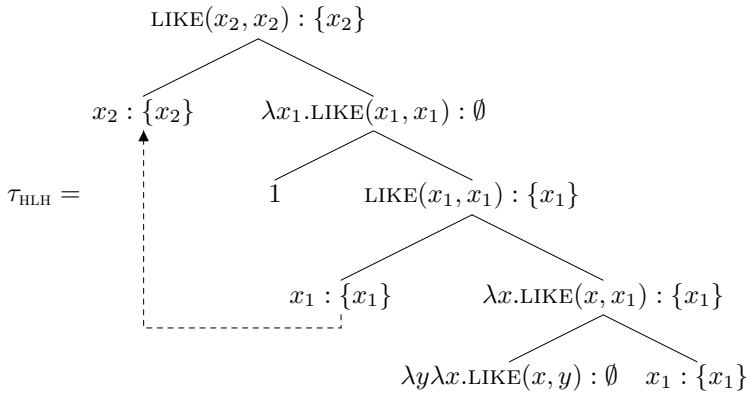
$\tau_{\text{LH}} =$



$\tau_{\text{H}} = x_2 : \{x_2\}$

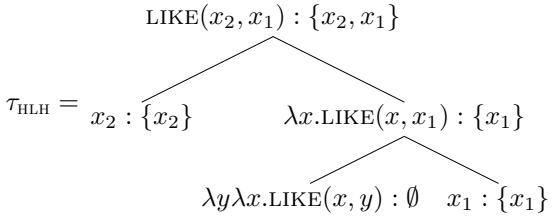
When τ_{LH} is combined with τ_{H} a configuration like the one described at point 2(a)i is detected: *he* and *him* have compatible agreement features, the former c-

commands the latter whose semantic interpretation is a variable. Therefore QR applies over *he* to get the following parse tree:



Actually this derivation contains the term $\lambda x_1.\text{LIKE}(x_1, x_1)$ which is a reflexive argument not reflexive-marked: the violation of Constraint 1 makes this derivations unacceptable, and the corresponding semantic interpretation $[\lambda x_1.\text{LIKE}(x_1, x_1)](x_2)$ forbidden, as well as it's $\beta-$reduction $\text{LIKE}(x_2, x_2)$. In this case it's reasonable to consider as forbidden readings all those which can be unified with $\text{LIKE}(x_2, x_2)$.

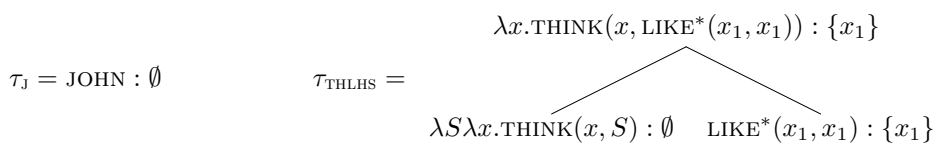
The computation for the logical form of the sentence goes on without QR to generate the following parse tree:



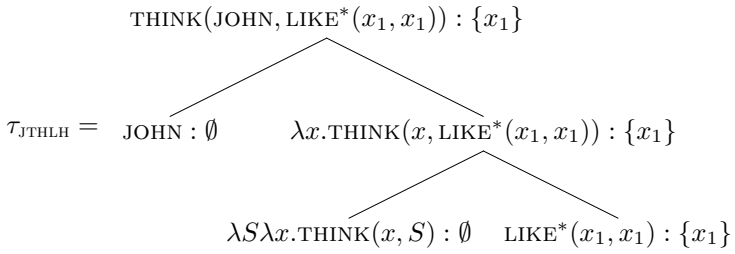
Therefore $\text{LIKE}(x_2, x_1)$ is the correct semantic representation for the sentence. Assignments like $x_2 \rightarrow x_1$ or $x_2 \rightarrow x_1$ are forbidden, as well as all those which have the effect of making the two arguments equal because they're all equivalent, modulo unification, to logical form $\text{LIKE}(x_2, x_2)$ which has already been identified as forbidden.

(42) *s*: John thinks that he likes himself.

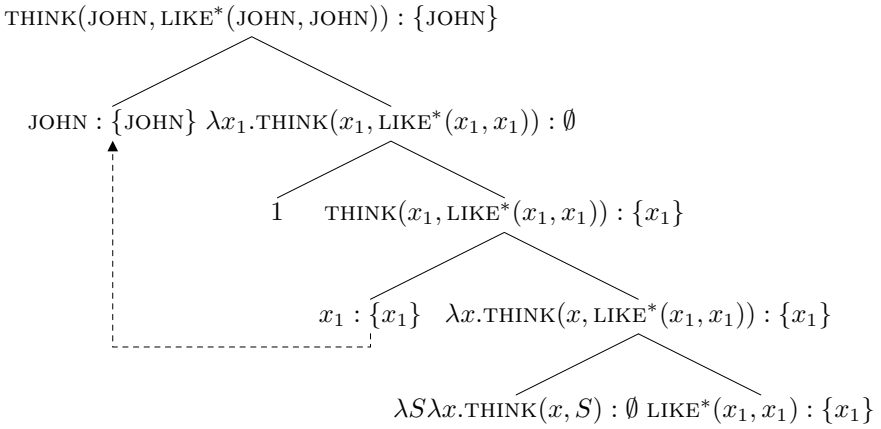
For simplicity, we assume that the algorithm successfully computes the semantics of the sentence *he likes himself* as the logical form $\text{LIKE}^*(x_1, x_1)$ (see example (38)). Then the resulting semantics for the whole sentence is given by the functional application between the logical forms corresponding to the fragments *John* and *thinks that he likes himself*, that is τ_{J} and τ_{THLHS} respectively:



When the semantic interpretation process gets to compute $\tau_{\text{J}} \cdot \tau_{\text{THLHS}}$ it is in the configuration described at point 2(a)i. So one additional reading is generated by means of QR (with x_1 copied in the trace position), and the resulting semantics is given by the following two logical form trees:



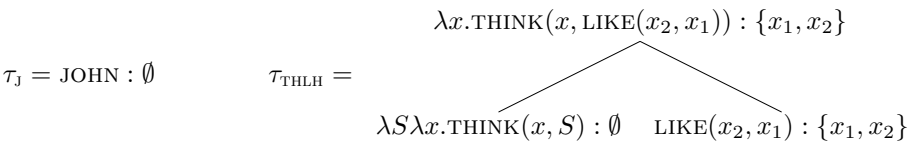
which corresponds to a reading in which *John* and *he* can be different individuals and there's no semantic binding between the corresponding linguistic entities, and τ'_{JTHLH}:



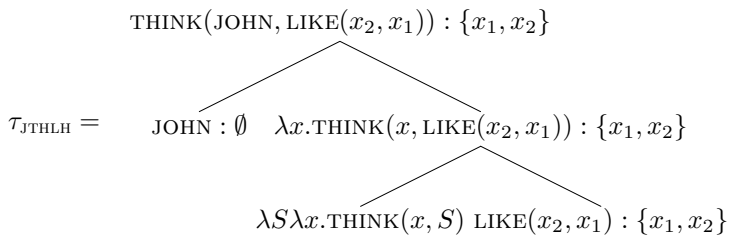
in which JOHN has been Quantifier Raised leaving x_1 as its trace, getting the reading in which *John* and *he* are semantically bound. As it can be easily verified, in no case there is an occurrence of a marked predicate in which at least two of its arguments are not identical.

(43) *s*: John thinks that he likes him

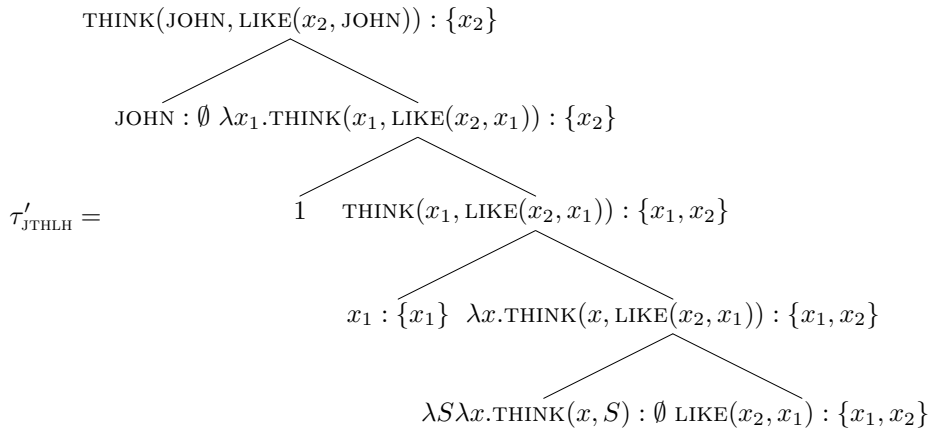
From example (40), we know that $\text{LIKE}(x_2, x_1)$ is the correct logical form for the fragment *he likes him*, while $\text{LIKE}(x_2, x_2)$, modulo unification, is forbidden. So in order to compute the semantic representation for the whole sentence it must compute that functional application between τ_{J} and τ_{THLH}:



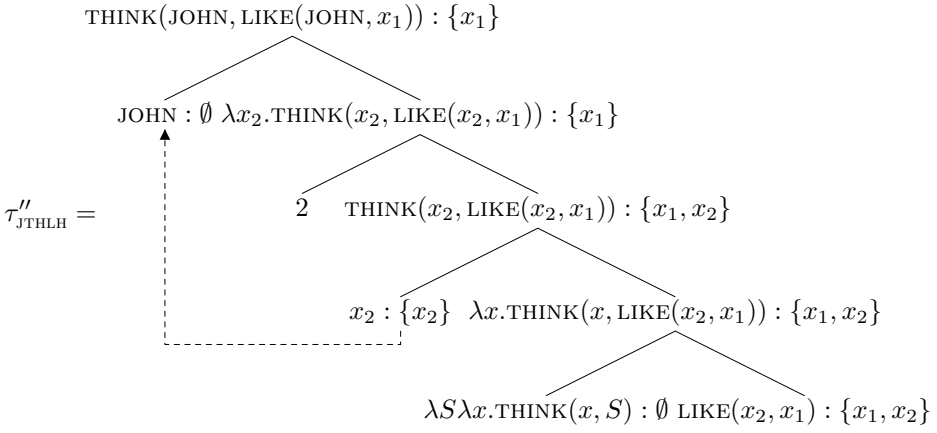
QR is triggered for each element of the set $\{x_1, x_2\}$ and two additional readings τ'_{JTHLH} and τ''_{JTHLH} are generated.



The logical form of τ_{JTHLH} corresponds to the reading in which no semantic binding occurs between *John*, *he* and *him*.



The logical form of τ'_{JTHLH}, in which JOHN has been Quantifier Raised and left x_1 as its trace, corresponds to the reading in which semantic binding occurs between *John* and *him*. There are no reflexive-marked predicates, and in no case a predicate gets two identical arguments.



The logical form of τ''_{JTHLH}, in which JOHN has been Quantifier Raised and left x_2 as its trace, corresponds to the reading in which semantic binding occur between *John* and *he*. There are no reflexive-marked predicates, and in no case a predicate gets two identical arguments.

4.5.9 Further linguistic data

The approach we presented in section 4.5.7 seamlessly extends to more complicated cases than the few elementary ones presented in the previous section. Grounded as it is on a very advanced interpretation for Binding Theory, it takes advantage of the linguistic coverage attained. We present how our algorithm works in these cases by highlighting only the main points of the analysis, the technical details having already being presented in the previous section.

Control

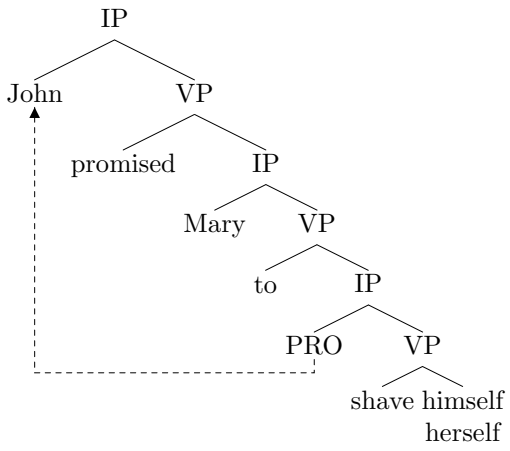
In the generative tradition, the term *control predicates* denotes words like *try*, *want*, *promise*, *persuade*, which take an infinitive complement with what is said to be a (controlled) PRO subject. This is the case of a sentence like (44):

(44) John promised Mary [PRO to shave himself]

The generative analysis of such a sentence assumes that the apparently subjectless bracketed infinitive clause contains an understood *null subject*, conventionally indicated as PRO (from *pronouns*, of which it is considered to have much the same grammatical and referential properties). PRO is an example of a *null* or *covert* category, i.e. a category which has no overt phonetic form but a tangible semantic content. The null PRO subject of the bracketed clause is said to be *controlled* by the subject *John* of the "next highest" clause. PRO acts as a referential pronoun that takes its reference from its controller, which must therefore be another referential expression. Otherwise stated, control phenomena only constrain assignment functions over independent linguistic entities. We ground our semantic computations on such kind of syntactic analysis. Consider the following contrast:

(45) a. John promised Mary [PRO to shave himself]
b. * John promised Mary [PRO to shave herself]

For which we adopt the following syntactic analysis



In compliance with our approach to control phenomena, we set $[\![PRO]\!] = x^{\text{CON}}$, to mark the special status of PRO as a pronoun which is neither free nor bound, but directly gets its reference from the controlling subject. Therefore, possible assignments to x^{CON} only depend on the mechanism ruling control phenomena, on which we do not make any assumption here.

When our bottom-up semantic computation performs the merge between *PRO* and *to shave himself*, it finds itself in the configuration already seen in example (38): the correct semantic reading generated for both the fragments *PRO to shave himself* and *PRO to shave herself* is therefore $[\lambda x_2.\text{SHAVE}^*(x_2, x_2)](x^{\text{CON}})$. We assume that the logical form of the predicate *promise* is $\lambda P \lambda y \lambda x.\text{PROMISE}(x, y, P(x))$. Therefore, both (45-a) and (45-b) are predicted to have the logical form:

(46) $[\lambda x.\text{PROMISE}(x, \text{MARY}, [\lambda x_2.\text{SHAVE}^*(x_2, x_2)](x^{\text{CON}}))](\text{JOHN})$

The mechanism that rules control resolution will assign to x^{CON} the semantic content of the controlling subject *John*, thus resulting in

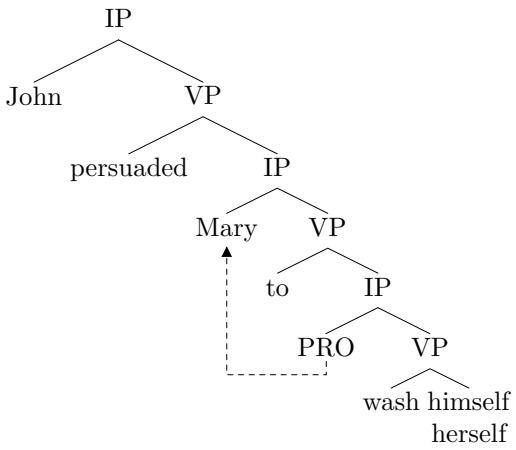
(47) $[\lambda x.\text{PROMISE}(x, \text{MARY}, [\lambda x_2.\text{SHAVE}^*(x_2, x_2)](\text{JOHN}))](\text{JOHN})$

which correctly reduces to $\text{PROMISE}(\text{JOHN}, \text{MARY}, \text{SHAVE}^*(\text{JOHN}, \text{JOHN}))$ in the case of (45-a), while for (45-b) it raises a agreement mismatch error between JOHN and x_2 which is supposed to have feminine gender features.

Control is not limited to subjects: verbs like *persuade* or *want* are said to be *object-control predicates*. The reason is immediately evident from the following contrast, entirely symmetrical to the previous one:

(48) a. * John persuaded Mary [PRO to wash himself]
b. John persuaded Mary [PRO to wash herself]

for which we adopt the following analysis:



We assume that the object-control predicate *persuade* has the logical form $\lambda P \lambda y \lambda x.\text{PERSUADE}(x, y, P(y))$. Therefore our semantic computations for (48-a) and (48-b) are the same as for, respectively, (45-a) and (45-b). However, the computation fails on (48-a) because this time it's the object *Mary* that must saturate

a position in the reflexive predicate *wash himself* with which it has incompatible agreement features. This is not the case in (48-b), for which only the logical form PERSUADE(JOHN, MARY, WASH*(MARY, MARY)) is correctly computed.

Raising

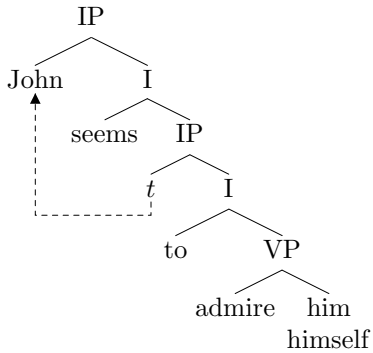
It is interesting to analyze how our algorithm behaves with another set of linguistic data represented by so-called *raising predicates*. Although superficially similar to control predicates they are substantially different at a deeper level of analysis. The analysis in this case is more complicated than in the case of control, in which the standard assumption is that control only constrains assignments over independent entities. The typical example of raising predicate is the verb *to seem*:

(49) a. It seems [that *he* understands her]
b. *He* seems [to understand her]

The puzzling evidence that the syntactic theory must account for is that the italicized expression which functions as the subject of the bracketed clause in (49-a) surfaces as the subject of the matrix clause in (49-b). The answer which is currently provided by generative approaches to syntax is that the italicized subject originates as the subject of the complement clause and is then *raised up* to become the matrix-clause subject by application of *raising* (not unlikely from what seen in section 2.2.7), leaving behind an empty trace as the subject of the complement clause. The subject and the trace are called the *head* and *tail* of a *chain*. Such a phenomenon is therefore entirely different from what happens with control verbs: in control phenomena, all the entities are independent and control only constrains assignment functions, while in raising phenomena we have a DP which is a composed object made of two linguistic elements, a head and a tail. Our semantic computations must therefore rely on an entirely different syntactic analysis, in particular one which involves raising and traces. Consider the following sentences:

(50) a. John seems to admire him
b. John seems to admire himself

The standard analysis which is proposed for such kind of sentences goes as follows:



The standard explanation for the construction of the sentence is that *John* has been generated in the trace position, and then is has undergone raising and

reached the final position as the matrix subject. The DP is not simply *John* but a binary object, a chain made of *John* and the trace t. Tail t cannot be considered as a DP. If we adhere to the "copy theory" for traces, we suppose that in t there is a copy of the semantic content of the head *John*. However, since the DP in this case is the whole chain, trace t cannot be quantifier-raised when merged with the fragment *admire him/himself*. If we assume that $[\![t]\!] = x^{\text{CH}}$ (where the special index marks the fact that the semantic content of the trace is a copy of the semantic content of the head of the chain), the only semantic representation computed for the fragment *seems t to admire himself* is $\lambda x.\text{SEEMS}(x, \text{ADMIRE}^*(x^{\text{CH}}, x_1))$. Being x^{CH} a special kind of variable, somewhat underspecified for the time being, this does not entail any violation of constraint 1 for the logical form.

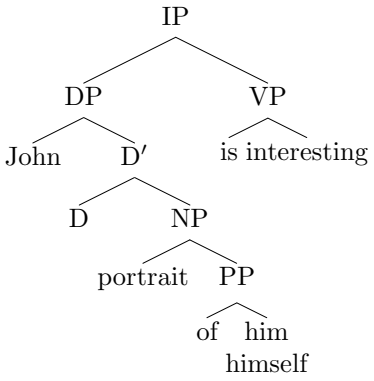
On the contrary, when the head of the chain *John* is merged with the fragment *seems t to admire himself*, Quantifier Raising applies, resulting in the logical form $[\lambda x_1.\text{SEEMS}(x_1, \text{ADMIRE}^*(x^{\text{CH}}, x_1))](\text{JOHN}^{\text{CH}})$.

Possessive DPs

Another interesting class of linguistic evidence that our approach can cope with is represented by constructions with possessive DPs like in the following contrast:

(51) a. John$_1$'s portrait of himself$_1$ is interesting
b. *John$_1$'s portrait of him$_1$

We assume the following generative analysis:



If we assume that *portrait* is a three-places predicate, when our algorithm gets to semantically merge DP *John* with the predicate *portrait of him*, it detects a Σ-configuration that triggers the Quantifier Raising of *John*. Ultimately, this leads to a semantic representation for the DP *John's portrait of him* as $[\lambda x_1 \lambda x.\text{PORTRAIT}(x, x_1, x_1)](\text{JOHN})$. Since we have a reflexive occurrence of a predicate which is not reflexive-marked in the syntax, this leads to an error and this reading is ruled out, as well as the result of its β-reduction $\lambda x.\text{PORTRAIT}(x, \text{JOHN}, \text{JOHN})$. The computation goes on with the free variable reading for pronoun *him*, yielding the logical form $\lambda x.\text{PORTRAIT}(x, \text{JOHN}, x_1)$. At this point, any assignment such that $x_1 \rightarrow \text{JOHN}$ will raise an error because such a reading has already previously computed as forbidden according to our unacceptability of derived readings rule.

On the contrary, when the personal pronoun is *himself*, the logical form computed for DP *John's portrait of him* is $[\lambda x_1 \lambda x.\text{PORTRAIT}^*(x, x_1, x_1)](\text{JOHN})$, which complies with reflexivity constraints and is therefore accepted.

4.5.10 Observations

In this section we presented an original computational treatment of Binding Theory which integrates the major insights of Reinhart [50] and Reinhart and Reuland [54].

The combination of features drawn from each of these theoretical approaches to Binding Theory was not an intellectual game. At the end of section 4.4 we had set a relatively simple machinery to generate all and only the correct logical forms of a sentence, but we had no clue on how to implement Reinhart's Rule I (whose standard formulation involve an instance of an NP-complete problem like SAT), and the issues of a suitable implementation of the notion of binding domain were still unaddressed. From a purely methodological point of view, our algorithm was for the large part an implementation of (binding) principles and stipulations drawn from linguistics. We might say that the "information" flowed only in one direction, from theoretical linguistics to computational linguistics.

In the present section we changed our perspective. While sticking to the original intuition of Reinhart [50] that binding theory is about semantic binding (implemented in section 4.4), by integrating insights taken from Reinhart and Reuland's Reflexivity we were able to provide an elegant treatment of Rule I issues. Our solution independently leads to a "division of labor" between syntax, semantics and pragmatics/discourse theory which naturally complies with the most recent linguistic approaches to Binding Theory.

Like any other, our approach stands on some stipulations: there is a natural "drive" for the interpretation process towards generating bound-variable readings for a sentence; there are interface conditions on the reflexive nature of predicates that block some of these configurations; such conditions can constrain the values (i.e. the assignment functions) of free pronouns in a sentence on the basis of logical forms that have been previously computed as unacceptable and which are stored somewhere in the memory of the speaker. However, we believe that all stipulations are quite natural, internally motivated instead of drawn from a set of rules defined theoretically[6]. Furthermore, they allow for an elegant interpretation process that can do away with binding domains, binding principles (in the traditional formulation), and in which c-command is an epiphenomenon of more primitive semantic operations. What was formerly an abstract structural condition is now

[6] Eric Reuland (private communication) draws our attention to the fact that our stipulated preference for bound-variable construals actually has experimental support. In a recent eye-tracking experiment by Arnout Koornneef at Linguistics Department of University of Utrecht it was found that under discourse conditions where a coreferent construal was pragmatically favored, and a bound-variable construal was implausible, but structurally possible, first a bound-variable construal was established followed by back-tracking. This nice piece of evidence seems to support, once again, our claim that algorithmic elegance can be a useful criterion to parallel, and sometimes anticipate, experimental results in linguistics.

naturally subsumed by the general structure of the algorithm and the bottom-up compositional approach.

The interaction between computational and theoretical linguistics worked the other way round: the set of principles and rules that we stated allow us to provide a better treatment of phenomena that needed a strong stipulation by Reinhart and Reuland (see section 4.5.6).

5

A Comparison with Schlenker's Approach

5.1 Schlenker's Fully Semantic Approach

Mainstream approaches to Binding Theory like Chomsky 1981, Reinhart 1983, Reinhart and Reuland 1993 all share a syntactic core. Coreferential and Reinhart's interpretations of Binding Theory basically consider it as part of the syntax: syntactic structures that would be interpretable by a semantic module are ruled out on the basis of purely formal (i.e. syntactic) constraints. Syntactic structures come equipped with indices associated to DPs whose intended semantics is to encode coreference or semantic binding. Chomsky's principles A, B and C are defined in terms of configurational relations holding between such indexed DPs. There is no semantic principle that forbids an interpretation of the sentence he_1 *likes* him_1 in which *he* and *him* happen to have the same denotation. It is the violation of a syntactic constraint (principle B) that rules out such a configuration, but ungrammatical structures are assumed to be in principle interpretable by the semantic component. Reinhart and Reuland's Reflexivity approach marks the return to a more semantics-based perspective on Binding Theory, but morphosyntax still plays a central role via the notion of reflexive-markedness of a predicate. In any case, there is no *semantic* impossibility to interpret ungrammatical sentences.

At the opposite end of the syntax-semantic spectrum lies the recent, fully semantic approach proposed by Philippe Schlenker (see [59], [58]). In his system syntactic well-formedness criterions for indexed structures are replaced by a semantic apparatus in which *denotational configurations* are ruled out on the basis of a single general principle of Non-Redundancy. Schlenker's approach is based on a cognitive metaphor: as a human speaker processes a sentence, he builds a memory register of the entities involved. Reflexive pronouns, non-reflexive pronouns, and full-DPs are processed as they occur in the sentence, and they modify the memory register according to a small set of basic rules. A general principle of Non-Redundancy must be satisfied: no object can occur twice in the memory register. On the basis of these purely semantic rules, Schlenker accounts for a wide range of phenomena without anything of the syntactic apparatus of familiar Binding Theory.

5.1.1 Schlenker's system overview

Schlenker's system is based on the idea of a *memory register* of the entities that occur in a sentence. The memory register is a metaphor for some kind of cognitive system which stores the denotations of the linguistic entities encountered while processing a sentence. When a new Determiner Phrase is encountered, the state of the register (described by a *sequence of evaluation*) is updated according to two rules:

- when a full-DP (proper name, definite description, demonstrative pronoun) is processed, its denotation is added at the end of the sequence of evaluation;
- when a non-demonstrative pronoun is processed, some element of the register is moved to the end of the register.

The first rule may be interpreted as the basic cognitive mechanism of storing in the last position of the memory register the most recent "cognitive file" corresponding to an individual. The values of individual denoting terms are added to the sequence of evaluation in an order that mirrors their hierarchy in the syntactic structure. In a top-down procedure, this order reflects the c-command relations that are found in the syntax, and this is the key to achieve a semantic reinterpretation of standard syntactic conditions on binding.

The second rule implements the idea that anaphoric pronouns do not introduce new "cognitive files" in the memory register, but they recover some previously stored denotation, making it available as an argument for the next predicate that's going to be processed (displacement to the last position of the evaluation sequence). Non-demonstrative pronouns come with negative indices associated, which indicate how far from the end of the register their denotations are to be found. Therefore he_{-1} evaluated under the sequence [JOHN; MARY; BILL] denotes BILL, he_{-2} denotes MARY, and he_{-3} denotes JOHN. This notation is similar to de Bruijn-style notation for lambda calculus, in which indices stand for the relative distance of the lambda operator that binds them (see [3] for details). Demonstrative pronouns are assumed to have positive indexes, and thus they do not recover their denotation from the sequence of evaluation but from the overall context the sentence is uttered within.

A sentence is interpreted by evaluating its components with respect to the sequence of evaluation. Non-demonstrative pronouns recover their denotations via the negative indexes that label them. The rules behind the construction of the sequence of evaluation guarantee that when we process a predicate P there is a mechanical procedure to recover its arguments from the evaluation sequence under which the predicate is evaluated. If P is an intransitive (i.e. one-place) predicate, the denotation of its only argument is to be found in the last position of the register; if P is transitive (i.e. binary), the denotations of its subject and object are in position -2 and -1 respectively. The truth value of a predicate P is always relative to a specific evaluation sequence s. If we indicate with I a suitable semantic interpretation function, the role of the sequence of evaluation in the interpretation process is formalized by the following rule:

Interpretation of predicates.

Let P be an n-place predicate, s a sequence of evaluation and s_n the sequence of the last n elements of s; then

$$[\![P]\!]s = \begin{cases} \# & \text{if and only if one of the last } n \text{ elements of } s \text{ is } \# \\ & \text{or } s \text{ violates Non-Redundancy} \\ s_n \in I(P) & \text{otherwise} \end{cases}$$

What follows is an example of a derivation in Schlenker's system (all examples are taken from [58] with minor notation changes). We indicate between square brackets the current state of the evaluation sequence:

$$\begin{aligned} [\![\text{John hates Bill}]\!]^s = 1 &\Leftrightarrow [\![\text{John hates Bill}]\!][s] = 1 \\ &\Leftrightarrow [\![\text{hates Bill}]\!][s; \text{J}] = 1 && \text{[Step 1: } \textit{John} \text{ is processed]} \\ &\Leftrightarrow [\![\text{hates}]\!][s; \text{J}; \text{B}] = 1 && \text{[Step 2: } \textit{Bill} \text{ is processed]} \\ &\Leftrightarrow (\text{J}, \text{B}) \in I(\textsc{hate}) && \text{[Step 3: } \textit{hates} \text{ is evaluated]} \end{aligned}$$

The subject and the object of the sentence are processed, and thus added to the sequence of evaluation, in an order which reflects the order in which they occur in the sentence. It is apparent that such order of introduction make c-commandees appear in the sequence of evaluation closer to the end that c-commanders. The last step in the evaluation process involves the interpretation of a binary predicate, which recovers the last two elements of the sequence of evaluation, which correctly happen to be the subject and the object of the predicate which must be interpreted.

Schlenker claims that on the basis of this simple formal semantic apparatus, a single principle that rules out redundancy in the sequence of evaluation suffices to account for most of the phenomena traditionally explained by traditional syntax-based approaches to Binding Theory:

Non-redundancy principle.

No object may occur twice in the same sequence of evaluation.

The whole point of Schlenker proposal is to prove that such a general semantic (or cognitive) principle alone, operating on the top of the simple semantic apparatus just sketched, accounts for a large part of phenomena that traditional Binding Theory deals with from a purely syntactic standpoint. Schlenker's principles are semantic in nature since they only deal with the way the sequence of evaluation are constructed and which evaluation sequences are acceptable according to the Non-Redundancy criterion. As a possible cognitive motivation for Non-Redundancy as a primitive mechanism of human mind, Schlenker hypothesizes a new kind of "economy principle" that states that a new cognitive file shouldn't be created for an object which is already stored in the memory of the speaker.

5.1.2 Non-demonstrative pronouns

The mechanism by which non-demonstrative (i.e. anaphoric) pronouns modify the sequence of evaluation asks for some additional technical apparatus. From the cognitive point of view of Schlenker's proposal, it's immediate to see why anaphoric pronouns neither leave the sequence of evaluation unchanged, nor they add new elements to it. Given that the most intuitive use of a pronoun in a sentence is to recover a previously introduced entity to access it as an argument of a predicate, and that in Schlenker's system arguments for predicates are consumed from the end of the sequence backward, it comes natural to state that a non-demonstrative pronoun *moves* a denotation already stored in the memory register and puts it at the end of the sequence which describes its state. To identify which element of the evaluation sequence must be displaced, non-demonstrative pronouns come with negative indexes associated. A pronoun with index $-i$ moves the denotation i-th element of a sequence of evaluation s from its actual position to the end of s. More formally:

Treatment of Non-Demonstrative Pronouns

If α is a pronoun with a negative index $-i$, $[\![\alpha\ \beta]\!]s = [\![\beta\ \alpha]\!]s = \#$ if s has fewer than i elements. Otherwise, for a possibly empty sequence s' and for some elements $d_1, \ldots, d_i$, $s = [s'; d_i; \ldots; d_1]$ and $[\![\alpha\ \beta]\!]s = [\![\beta\ \alpha]\!]s = [\![\beta]\!][s'; \#; d_{i-1}; \ldots; d_1; d_i]$.

A moved object leaves a blank element # behind it in the evaluation sequence. Such a technical device completes the technical implementation of the semantics of anaphoric pronouns. Processing a non-demonstrative pronoun not only makes a previously introduced denotation available as an argument for the next predicate, but also makes it unavailable to other predicates (the interpretation of a predicate with # as an argument is supposed to fail).

The treatment of non-demonstrative pronouns allow Schlenker's system to account for some basic cases of Principles B violation effects. What follows is an example of a derivation which fails not out of Non-Redundancy violation, but because the movement of the denotation induced by the anaphoric pronoun leaves a blank space which is not filled by any other argument for the binary predicate:

$$
\begin{aligned}
[\![\text{John likes him}_{-1}]\!]^s = 1 &\Leftrightarrow [\![\text{John likes him}_{-1}]\!][s] = 1 \\
&\Leftrightarrow [\![\text{likes him}_{-1}]\!][s; \textsc{j}] = 1 && \text{[Step 1: } \textit{John} \text{ is processed]} \\
&\Leftrightarrow [\![\text{likes}]\!][s; \#; \textsc{j}] = 1 && \text{[Step 2: } \textit{him}_{-1} \text{ is processed]} \\
&\Leftrightarrow (\#, \textsc{j}) \in I(\textsc{like}) && \text{[Step 3: } \textit{likes} \text{ is evaluated]}
\end{aligned}
$$

The last step, which entails that a predicate is evaluated with the # symbol as one of its arguments, makes the interpretation procedure fail, as desired.

5.1.3 Full-DPs and *that*-clauses

An additional rule is needed in Schlenker's system to account for phenomena traditionally involving c-command outside a local domain.

Treatment of full-DPs and *that*-clauses

If α is a proper name, a definite description, a demonstrative pronoun or a *that*-clause: $[\![\alpha\ \beta]\!]s = [\![\beta]\!][s; [\![\alpha]\!]s]$

Informally, we can say that every time the interpretation procedure gets to process an r-expression or a *that*-clause α, a local copy of the evaluation sequence is created, under which α is evaluated, and the result is added to the initial sequence.

In the following example we abstract from technical details needed to deal with the intensionality issues raised by the presence of the opaque context introduced by the predicate *claims* to concentrate on the inner workings of the memory register:

$$\begin{aligned}
[\![\text{Bill claims that he}_{-1}\text{ runs}]\!]^s = 1 &\Leftrightarrow [\![\text{claims that he}_{-1}\text{ runs}]\!][s; \text{B}] = 1 \\
&\Leftrightarrow [\![\text{claims}]\!][s; \text{B}; p] = 1 \\
\text{with } p &= [\![\text{that he}_{-1}\text{ runs}]\!][s; \text{B}] \\
&= \lambda w.[\![\text{he}_{-1}\text{ runs}]\!]^w[s; \text{B}] \\
&= \lambda w.[\![\text{runs}]\!]^w[s; \#; \text{B}] \\
&= \lambda w.\text{B} \in I_w(\text{RUN})
\end{aligned}$$

When the interpretation procedure gets to process the *that*-clause, the evaluation sequence is $[s; \text{B}]$. The computation stops to process the *that*-clause, which is evaluated under a copy of the same evaluation sequence, that is $[s; \text{B}]$. Non demonstrative pronoun he_{-1} can therefore correctly access DP *Bill* previously introduced in the main clause to be fed to predicate *runs*. The result p is then added to the evaluation sequence for the main clause.

The rule we just introduced for *that*-clauses naturally accounts for the following contrast, traditionally accounted for in terms of c-command accessibility:

(1) a. Bill's teacher likes Bill
b. *Bill likes Bill's teacher

If we assume (as it is the case in the large majority of contexts in which such a sentence can be uttered) that the two occurrences of Bill refer to the same individual, traditional Binding Theory would rule out (1-b) due to a violation of principle C (the second occurrence of full-DP *Bill* is bound the first one). This is not the case in (1-a), where no binding occurs because the first occurrence of *Bill* is embedded into the DP constituent *Bill's teacher* and thus does not c-command the second one. Schlenker's system accounts for this contrast on a purely semantic ground.

In (1-a) the constituent *Bill's teacher* is processed under an evaluation sequence s which does not contain any other element: the result (the denotation T) is the added to the evaluation sequence. Denotation T does not clash with the second occurrence of *Bill*, so no redundancy violation occurs and the sentence is deemed acceptable. Intuitively, the file which is stored in memory is about Bill's teacher, a whole different individual from Bill himself.

$$\begin{aligned}
[\![\text{Bill's teacher likes Bill}]\!]^s = 1 &\Leftrightarrow [\![\text{likes Bill}]\!][s; p] = 1 \quad (\textit{Bill's teacher} \text{ is processed}) \\
\text{with } p = &\ [\![\text{Bill's teacher}]\!][s] = [\![\text{teacher}]\!][s; \textsc{b}] = \textsc{t} \\
&\Leftrightarrow [\![\text{likes Bill}]\!][s; \textsc{t}] = 1 \\
&\Leftrightarrow [\![\text{likes}]\!][s; \textsc{t}; \textsc{b}] = 1 \\
&\Leftrightarrow (\textsc{t}, \textsc{b}) \in I(\textsc{like}) = 1
\end{aligned}$$

In (1-b) *Bill's teacher* is processed under an evaluation sequence s which is not empty like in the previous case, but which already contains the denotation of *Bill*.

$$\begin{aligned}
[\![\text{Bill likes Bill's teacher}]\!]^s = 1 &\Leftrightarrow [\![\text{likes Bill's teacher}]\!][s; \textsc{b}] = 1 \quad (\textit{Bill} \text{ is processed}) \\
&\Leftrightarrow [\![\text{likes}]\!][s; \textsc{b}; p] = 1 \\
\text{with } p = &\ [\![\text{Bill's teacher}]\!][s; \textsc{b}] = [\![\text{teacher}]\!][s; \textsc{b}; \textsc{b}]
\end{aligned}$$

The sequence of evaluation for the full-DP *Bill's teacher* is evaluated under an extension of sequence $[s; \textsc{b}]$ which has been created after the first occurrence of *Bill* has been processed in the main clause. Therefore, it contains two occurrences of the same entity, and this brings about a Non-Redundancy violation thus resulting in an unacceptability judgement for the whole sentence.

5.1.4 Condition C effects

The combination of the Non-Redundancy principle and the machinery we have described so far to interpret *that*-clauses and full-DPs allows Schlenker's system to account for a large number of cases that are dealt with by principle C of traditional Binding Theory. The following example shows how Schlenker's system deals with basic occurrences of Principle C violations:

$$\begin{aligned}
[\![\text{John likes John}]\!]^s = 1 &\Leftrightarrow [\![\text{John likes John}]\!][s] = 1 \\
&\Leftrightarrow [\![\text{likes John}]\!][s; \textsc{j}] = 1 \qquad [\text{Step 1: } \textit{John} \text{ is processed}] \\
&\Leftrightarrow [\![\text{likes}]\!][s; \textsc{j}; \textsc{j}] = 1 \qquad [\text{Step 2: } \textit{John} \text{ is processed}]
\end{aligned}$$

Non-redundancy principle forbids two occurrences of the same denotation in a sequence of evaluation, and thus the interpretation procedure fails as desired. We would get the same effect in case the subject of the sentence was a demonstrative pronouns like he_1 such that $[\![he_1]\!] = \textsc{j}$.

In the classical formulation of Binding Theory, a sentence like he_1 *likes* him_1 is ruled out by principle B because it displays a syntactic configuration which involves the binding of a non-reflexive pronoun within its local domain. Schlenker's system justifies its unacceptability on the ground of purely semantic considerations. Since both *he* and *him* are supposed to refer to the same object, the sequence of evaluation will store two occurrences of the same denotation, which leads to a violation of the Non-Redundancy principle.

5.1.5 Condition A effects

Reflexive pronouns yield local coreferential readings, which are forbidden out of Non-Redundancy violations for non-reflexive pronouns. On the other hand, it is apparent that reflexive pronouns do not actually introduce new entities into the evaluation sequence. Therefore, reflexive pronouns need a different formal apparatus, and Schlenker justifies their deviant behavior by considering them as arity-reducer operators.

Treatment of reflexive pronouns

Let P be an n-place predicate, s a sequence of evaluation and s_1 the last element of s; then

$$[\![self{-}P]\!]s = \begin{cases} \# & \text{iff } s = \# \text{ or } s \text{ violates Non-Redundancy} \\ (s_1, s_1) \in I(P) & \text{otherwise} \end{cases}$$

What follows is an example of a derivation in Schlenker's system involving reflexive pronouns:

$$\begin{aligned} [\![\text{Bill hates himself}]\!]^s = 1 &\Leftrightarrow [\![\text{Bill self-hate}]\!][s] = 1 \\ &\Leftrightarrow [\![\text{self-hate}]\!][s; \text{B}] = 1 \\ &\Leftrightarrow (\text{B}, \text{B}) \in I(\text{HATE}) \end{aligned}$$

In order to face typical problems of the arity-reducer approach, namely with predicates which allow for more than one local antecedent, Schlenker integrates this approach by considering reflexives likes *himself* as composed of two parts:

- *him*, which behaves like a non-demonstrative pronoun, and moves an element of the sequence to its final position, leaving behind an empty cell. This has also the advantage of making the displaced denotation unavailable for further anaphoric usage, as required.
- *self* as an arity-reducing operator, which turns an n-place predicate into a (n-1)-place predicate. In order to specify which position is reflexivized, the notation $SELF_{i/k}$ is introduced, to indicate that the i-th and k-th position of the predicate will be coreferential.

Therefore Schlenker proposes the following derivation for the sentence *Ann introduced Berenice to herself*, where *herself* is supposed to refer to *Ann*:

$$\begin{aligned} &[\![\text{Ann [Berenice [SELF}_{1/3}\text{-introduce her}_{-2}]]\!]^s = 1 \Leftrightarrow \\ &\Leftrightarrow [\![\text{Berenice [SELF}_{1/3}\text{-introduce her}_{-2}]\!][s; \text{A}] = 1 \ (\textit{Ann} \text{ is processed}) \\ &\Leftrightarrow [\![\text{SELF}_{1/3}\text{-introduce her}_{-2}]\!][s; \text{A}; \text{B}] = 1 \ (\textit{Berenice} \text{ is processed}) \\ &\Leftrightarrow [\![\text{SELF}_{1/3}\text{-introduce}]\!][s; \#; \text{B}; \text{A}] = 1 \ (\textit{her}_{-2} \text{ is processed}) \\ &\Leftrightarrow (\text{A}, \text{B}, \text{A}) \in I(\text{INTRODUCE}) \ (\text{SELF}_{1/3}\text{-introduce is evaluated}) \end{aligned}$$

As Schlenker himself points out, this analysis runs into some technical problems. For example, a sentence like *Ann introduced Berenice to herself*$_{-1}$, where *herself* is supposed to corefer with *Berenice* would wrongly raise a semantic failure. Schlenker must resort to a more sophisticated version of his system, involving temporal anaphora, to cope with this problem.

5.1.6 Quantification

Schlenker's system faces a serious problem with respect to quantificational expressions. The problem is addressed by Schlenker himself and exemplified by the sentence *Ed thinks that every professor is underpaid*. By following the standard assumption that traces behave like non-demonstrative pronouns, the fragment *every professor is underpaid* should be analyzed in Schlenker's system as follows:

$$\begin{aligned}
&[\![\text{[every professor] [t}_{-1}\text{ is underpaid]}]\!][s] = 1\\
\Leftrightarrow\ &\forall \text{D.D} \in I(\textsc{professor}), [\![t_{-1}\text{is underpaid}]\!][s; \text{D}] = 1\\
\Leftrightarrow\ &\forall \text{D.D} \in I(\textsc{professor}), [\![\text{is underpaid}]\!][s; \#; \text{D}] = 1\\
\Leftrightarrow\ &\forall \text{D.D} \in I(\textsc{professor}), \textsc{underpaid}(\text{D}) = 1
\end{aligned}$$

The problem is that if Non-Redundancy is checked with respect to each of the many sequences that enter in the truth-conditions, it predicts that *Ed thinks that every professor is underpaid* cannot mean that Ed thinks that every professor *including himself* is underpaid. In fact, once that *Ed* has been processed at the beginning of the interpretation process, it will occur in any sequence of evaluation derived from the main one: in particular for $d = [\![Ed]\!] = \text{E}$, two occurrences of E will appear in the evaluation sequence and Non-Redundancy condition is violated. This means that the interpretation procedure fails under an interpretation in which Ed thinks that he himself is underpaid.

Schlenker is forced to complicate his apparatus with an ad hoc stipulation to deal with quantification to account for such situations. The additional machinery implements the intuitive idea that the elements introduced by a quantifier do not appear in the sequence of evaluation but in a *quantificational sequence* which is not subject to Non-Redundancy. Its elements must be able to access the denotations occurring in the main sequence, and this is made possible by traces which are thus endorsed with a much different role from non-demonstrative pronouns. The positive side of this complication of the formal apparatus is that is succeeds also in accounting for Weak and Strong Crossover effects. This mechanism is based on the two steps of Introduction and Cross-reference:

Introduction step: when a quantifier is evaluated with respect to a sequence of evaluation s and a quantificational sequence q, it leaves s unchanged but turns q into $[q; d]$ for each object d that is quantified over;

Cross-reference step: when a trace indexed with the quantifier is processed, an index i is introduced in the sequence of evaluation which indicates which cell of the quantificational sequence must be retrieved.

This means that a quantificational statement like *every professor is underpaid* would be analyzed as follows:

$$\begin{aligned}
&[\![\text{[every professor] [}t_{-1}\text{ is underpaid]}]\!][s] = 1 \Leftrightarrow\\
\Leftrightarrow\ &\forall x \in X \text{ satisfying } [\![\text{professor}]\!][s][x = 1], [\![t_{-1}\text{is underpaid}]\!][s][x = 1]\\
\Leftrightarrow\ &\forall x \in X \text{ satisfying } x \in I(\textsc{professor}), [\![\text{is underpaid}]\!][s; 1][x = 1]\\
\Leftrightarrow\ &\forall x \in X \text{ satisfying } x \in I(\textsc{professor}), ([s; 1]_1), x \in I(\textsc{underpaid})\\
\Leftrightarrow\ &\forall x \in X \text{ satisfying } x \in I(\textsc{professor}), x \in I(\textsc{underpaid})
\end{aligned}$$

5.2 Truth-conditional and Denotational Economy

In section 3.3 we have seen how Reinhart resorts to introduce a notion of *truth-conditional economy* to select configurations which involve semantic binding over those which do not in any situation in which this does not change the truth condition of the sentence.

The same principle of economy can be applied with respect to the problem of *locality of variable binding*. Any theory in which binding is a non-transitive relation between two expressions has to provide, in addition with a version of classical Principle B, according to which two DPs cannot be bound locally, a principle that *requires* local binding in certain configurations. Consider the following examples, in which the arrows stand for binding relations between DPs:

(2) a. Bill claims that he hates him

b. Bill claims that he hates him

Binding configuration (2-a) is ruled out in traditional approaches to Binding Theory by a violation of principle B, which forbids local binding between *he* and *him*. However, as far as binding principles go, nothing forbids configuration (2-b), since no local binding of non-reflexive pronouns occurs. An additional principle known as "Locality of Variable Binding" must be introduced to block such configuration. Intuitively, configuration (2-b) is ruled out because in it *him* is bound non-locally by *Bill* even though local binding by *he* would yield the same semantic result. Therefore a principle of local binding must ensure that if the semantic content of the sentence is not affected, then local variable binding configurations must be chosen over non-local ones.

In order to explain this effect, a notion of *economy* must be introduced. The general idea is that local binding is more economical, and thus preferred, than non-local binding. However, there are at least two ways to interpret the notion of economy. Truth-conditional economy (see [50], [25], [20]) states that local binding must be preferred unless non-local binding yield a logical form with different *truth conditions*. By contrast, Kehler in [34] argues for a weaker, but more computationally appealing principle of *denotational economy*, which requires local binding unless non-local binding yields a different *denotation* for the bound pronoun. More precisely:

Truth-conditional economy: for any two DPs α and β, if α could bind β (i.e. if α c-commands β, and β is not already bound in α's c-command domain already), α must bind β, *unless this changes the truth conditions of the entire sentence.*

Denotational economy: for any two DPs α and β, if α could bind β (i.e. if α c-commands β, and β is not already bound in α's c-command domain already), α must bind β, *unless this changes the denotation of* β.

Schlenker proves that his system, without any further machinery, predicts that denotational economy is always satisfied. Otherwise stated, denotational economy does not come as an additional rule operating on top of a previous analysis to filter out incorrect readings, but its effects naturally stem from the overall architecture of his system. This can be easily verified by considering the analysis that Schlenker's system provides for the sentences in (2). Binding configurations in (2-a) and (2-b) would translate in Schlenker's notation as follows, respectively:

(3) a. Bill claims that he_{-1} hates him_{-1}
b. Bill claims that he_{-1} hates him_{-2}

For each of the two binding configurations, after he_{-1} has been processed the evaluation sequence is $[s; \#; \text{B}]$. So when him_{-1} is processed in (3-a), the evaluation sequence turns into $[s; \#; \#; \text{B}]$, and predicate *hates* is evaluated with # as one of its arguments, thus making the evaluation fail. In (3-b), processing him_{-2} modifies evaluation sequence into $[s; \#; \text{B}; \#]$: once again the predicate gets evaluated with a # symbol in one of its argument positions. This confirms how the effects of denotational economy are implicit in the general architecture of Schlenker's system.

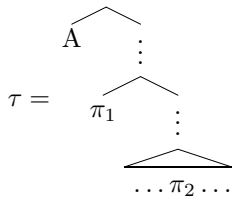
5.3 Comparison with Schlenker's approach

The integrated computational approach we presented in chapter 4 and Schlenker's proposal move from very distant, and somewhat contradictory, starting points. While the main guidelines for our treatment of Binding Theory are computational efficiency and algorithmic elegance, Schlenker's work stems from a purely speculative intent to provide a new and psycholinguistically plausible explication of why in Binding Theory things are the way they are. It is interesting to compare some of the basic features of both approaches.

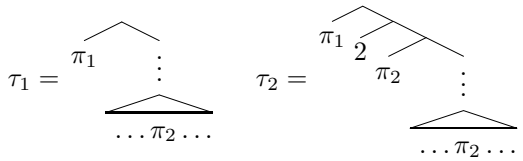
Denotational Economy

As pointed out in section 5.2, Schlenker proves that his system predicts that denotational economy should always be satisfied. Otherwise stated, Kehler's principle does not need to be stipulated to rule out undesired binding configurations like (2-b), but descends seamlessly from the overall architecture of Schlenker's system. By walking on Schlenker's footprints, we can show that the very same property holds in our system.

Let's consider a sentence in which there are three coreferential expressions like $A \ldots \pi_1 \ldots \pi_2$, where A is a DP which c-commands π_1, π_1 c-commands π_2, and π_1 and π_2 are both non-reflexive pronouns. The corresponding phrase-marker has the structure of τ:

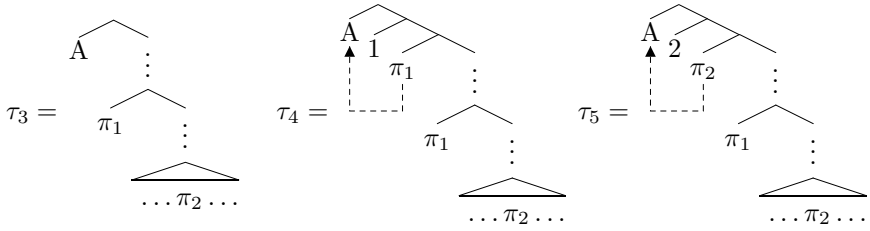


This is the configuration of a sentence like *John*$_1$ *claims that he*$_1$ *believes that he*$_1$ *is smart*. Denotational economy predicts that the configuration in which *John* binds both the first and the second occurrence of *he*$_1$ is not allowed. Our algorithm generates two logical forms for the fragment *he believes that he is smart*, one which does not involve semantic binding and one which does, which correspond to τ_1 and τ_2 respectively. For the latter, coreference is a consequence of semantic binding, in the former it can be achieved only in contexts such that $[\![\pi_1]\!] = [\![\pi_2]\!]$.



If we assume $[\![\pi_1]\!] = x_1$ and $[\![\pi_2]\!] = x_2$, phrase-markers τ_1 and τ_2 correspond to the logical forms $\textsc{believe}(x_1, \textsc{smart}(x_2))$ and $[\lambda x_2.\textsc{believe}(x_2, \textsc{smart}(x_2))](x_1)$, respectively.

When τ_1 is semantically merged with the DP constituent A, the algorithm generates three possible logical forms: (i) one in which no semantic binding occurs between the DPs; (ii) one in which, due to QR of A, A and π_1 get semantically bound; (iii) one in which, due to QR of A, A and π_2 get semantically bound. They correspond to phrase-markers τ_3, τ_4 and τ_5, respectively:

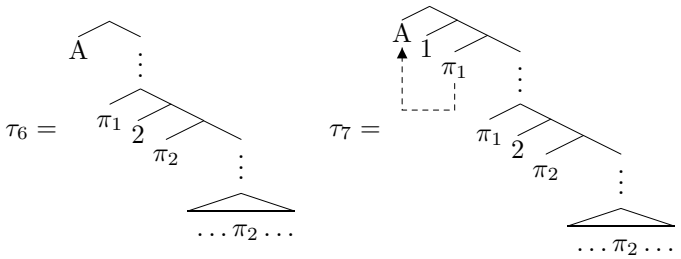


Under the usual assumptions, the logical forms for τ_1, τ_2 and τ_3 are, respectively:

(4) a. $\textsc{claim}(\textsc{john}, \textsc{believe}(x_1, \textsc{smart}(x_2)))$
b. $[\lambda x_1.\textsc{claim}(x_1, \textsc{believe}(x_1, \textsc{smart}(x_2)))](\textsc{john})$
c. $[\lambda x_2.\textsc{claim}(x_2, \textsc{believe}(x_1, \textsc{smart}(x_2)))](\textsc{john})$

None of the previous logical forms involve a configuration in which simultaneous binding occur between A and π_1 and π_2.

When τ_2 is merged with DP constituent A, only two logical forms are generated. Lambda-abstraction over π_2 makes it unavailable for further binding, and therefore we get (i) a logical form in which there is no binding between A and π_1 and (ii) a logical form in which semantic binding occurs as a consequence of QR applied to A; they correspond to τ_6 and τ_7, respectively:



Tree-markers τ_6 and τ_7 correspond to the following logical forms, respectively:

(5) a. $\textsc{claim}(\textsc{john}, [\lambda x_2.\textsc{believe}(x_2, \textsc{smart}(x_2))](x_1)) \rightarrow$
$\rightarrow \textsc{claim}(\textsc{john}, \textsc{believe}(x_1, \textsc{smart}(x_1)))$

b. $[\lambda x_1.\textsc{claim}(x_1, [\lambda x_2.\textsc{believe}(x_2, \textsc{smart}(x_2))](x_1))](\textsc{john}) \rightarrow$
$\rightarrow [\lambda x_1.\textsc{claim}(x_1, \textsc{believe}(x_1, \textsc{smart}(x_1)))](\textsc{john}) \rightarrow$
$\rightarrow \textsc{claim}(\textsc{john}, \textsc{believe}(\textsc{john}, \textsc{smart}(\textsc{john})))$

Once again, no logical form generated involve simultaneous semantic binding by A over both π_1 and π_2, as desired.

On the other hand if we consider a sentence like *John claims that he hates him* the analysis provided for example (43) shows that no semantic binding occurs in which *John* binds both *he* and *him*. Our system, just like Schlenker's, does not need any additional Locality of Variable Binding principle to rule out configurations like (2-b).

From example (43) of section 4.5.8 we know that for the sentence *John thinks that he likes him* our system generates three possible logical forms corresponding to as many semantic readings:

(6) a. $\textsc{claim}(\textsc{john}, \textsc{like}(x_1, x_2))$

b. $[\lambda x_1.\textsc{claim}(x_1, \textsc{like}(x_1, x_2))](\textsc{john}) \stackrel{\beta}{\Longrightarrow} \textsc{claim}(\textsc{john}, \textsc{like}(\textsc{john}, x_2))$

c. $[\lambda x_2.\textsc{claim}(x_2, \textsc{like}(x_1, x_2))](\textsc{john}) \stackrel{\beta}{\Longrightarrow} \textsc{claim}(\textsc{john}, \textsc{like}(x_1, \textsc{john}))$

Furthermore, the system generates a constraint on the assignment function that forbids any assignment that unifies the logical form corresponding to *he likes him* with $\textsc{like}(x, x)$. Therefore it is never then case that simultaneous binding occurs between *John* and *he* and between *John* and *him*. Either *John* binds *he* or *John* binds *him*: in either case, the constraint on the assignment function prevents the two pronouns from getting the same denotation. This means that the effects of the Locality of Variable Binding are automatically included in our system without any further stipulation as it is the case for Schlenker's system.

Psycholinguistic plausibility

Schlenker's system presents a very straightforward and psycholinguistically plausible approach to Binding Theory. The notion of a memory register in which entities are stored as they get processed by a human speaker, and which prevents him, on the basis of an economy principle, from introducing the same entity more than once is both intuitive and effective in accounting for several phenomena dealt with

at a syntactic level by more traditional approaches to Binding Theory. Anaphoric pronouns are seen as a linguistic device that allows a speaker to access previously introduced entities whenever needed without violating this general principle of cognitive economy. In doing so, Schlenker gets rid of most of the syntactic apparatus of traditional approaches to Binding Theory which deals with locality domains and c-command. Locality issues are captured by the relatively mild stipulations on the treatment of full-DPs and *that*-clauses that we presented in section 5.1.3. The structural notion of c-command disappears in Schlenker's system: c-command bears no primitive role in his theory, being no more than an *a posteriori* configurational description of a genuinely semantic effect of the general architecture of the system, issued from the level of embedding of a linguistic entity within a constituent.

Although our integrated approach stems from a concern for computational efficiency and algorithmic elegance more than from psycholinguistic plausibility, we come to somewhat similar conclusions in rejecting c-command as a primitive notion. As in Schlenker's system, c-command bears no role by itself, being subsumed by the general inductive architecture of the system and by the overall semantic perspective we assume on Binding Theory.

Quantification

Schlenker gets rid of all the syntactic apparatus of traditional approaches to Binding Theory by assuming a down-to-earth, intuitive approach to pronouns and entities. However, his system falls short with respect to the very delicate issue of quantificational expressions. By stripping semantic theory of all of its syntactic apparatus, and thus giving up to the notion of binding as a whole, Schlenker faces a problem in every situation in which semantic binding provides an elegant an effective way to deal with the omnipresent phenomena of quantification. Therefore he is forced to introduce admittedly heavy further stipulations and formal machinery to deal with it. Our approach is totally transparent to quantification issues. The central notion of our system always being semantic binding, we take advantage of the major intuitions and achievements issued from Reinhart's approach on this issue and we transparently integrate them into a computational framework. No further machinery is needed beyond the usual assumptions on covert movement of quantificational expressions which are common in formal semantics. Our approach deals with traces in exactly the same way it deals with non-reflexive pronouns. In the case of sentence *Ed thinks that every professor is underpaid*, that causes so much trouble to Schlenker's system, no problem arises for ours. Given the semantic analysis as [Every professor][t_1 is underpaid] (where the quantificational DP has been Quantifier Raised leaving trace t_1 in the original position), the constituent [t_1 is underpaid] will receive the usual semantic analysis as $\lambda x.\textsc{underpaid}(x)$ (not differently from the sentence *he is underpaid*). Usual semantic for quantificational DP will apply to get the final semantic representation as $\forall x.[\textsc{professor}(x) \rightarrow \textsc{underpaid}(x)]$ for the fragment *every professor is underpaid*. When later on during the inductive process type e DP Ed gets merged with *every professor is underpaid*, no new c-command configuration gets established since the trace has already been abstracted over and therefore cannot enter

a semantic binding relation with *Ed* anymore. Therefore our system does not incur in any of the problems that force Schlenker to create additional formal machinery to deal with quantificational expressions.

Coreference and binding

Another important consequence of Schlenker radical approach lies in the fact that in the present theory the strict/sloppy distinction cannot be represented syntactically, because for any coreferential reading involving c-command there is a single reading that can represent it. As Schlenker himself points out, in his system the only possible way to analyze a sentence like *John thinks that Mary likes him* is by associating index -2 to *him*, thus making it coreferential with *John*. In our system, two readings are possible, and are actually generated by the algorithm, namely the two following logical forms:

(7) a. $[\lambda x.\textsc{think}(x, \textsc{like}(\textsc{mary}, x))](\textsc{john})$
b. $[\textsc{think}(\textsc{john}, \textsc{like}(\textsc{mary}, x))]$

There is no constraint in (7-b) that prevents x from being mapped into JOHN, therefore getting a truth-conditionally equivalent reading to the sentence which does not involve semantic binding. Otherwise stated, in our system there are still two ways by which *him* might come to refer to *John*, either via a bound-variable approach or via a coreferential reading of a free variable. Schlenker's approach makes only the first one available, while coreferential reading between *John* and *him* is deemed impossible: if *him* has to be considered as a demonstrative pronoun such that $[\![him]\!] = \textsc{john}$, then it would be evaluated under an evaluation sequence that already contains JOHN and thus the computation would fail because of a Non-Redundancy violation.

In our approach the strict/sloppy contrast can still be represented syntactically, due to the two different readings that are generated. Although Schlenker claims that this shortcoming of his approach is not really such since the ambiguity theory cannot account for all the empirical facts, it is a matter of fact that an approach which saves most of current hypothesis about the strict/sloppy identity puzzle has an empirical edge over one which does not.

Principles B and C

One of the most remarkable achievements of Schlenker's approach is in our opinion the very natural way by which it succeeds in unifying effects which were formerly classified as Principle C or principle B violations, thus providing a convincing common explanation of their common obviative character. Sentences like he_1 *likes* $John_1$, $John_1$ *likes* him_1, $John_1$ *likes* $John_1$ all fail for the same reason, that is resorting to linguistic expressions that introduce twice the same entity in the memory register, thus entailing a violation of Non-Redundancy principle.

We consider our approach to be slightly less successful under this respect. Traditional principle B violations effects are a consequence of the violation of an interface filter which rules out reflexive readings for predicates which are not reflexive-marked. In order to achieve principle C obviative effects instead we had

to introduce an additional principle of "lexical obviation" in section 4.5.5. In some contexts, this lead to some redundancy (see the observation at the end of section 4.5.5). However, we consider that our principle is motivated on empirical terms, and, beyond intrinsic elegance, there are no compelling reasons to postulate a common root for principle B and C obviative effects. On the other hand we consider our additional machinery an acceptable price for a system which does not run into the problems of Schlenker's with respect to quantification.

Principle A and logophoricity

Schlenker must resort to the old idea of reflexive pronouns as arity-reducers operators to account for principle A effects. Our approach is not entirely different, although the reflexive nature of a predicate is a combined effect of Quantifier Raising blindly applied to DPs which get merged with constituents which contain other DPs, and interface filters which rule out reflexive readings for predicate which are not reflexive-marked. However, our general approach to reflexive pronouns allows to account for logophoric use of anaphors as well, while Schlenker's system at the present stage of development does not address this issue.

6

Conclusions

We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time.

T.S. Eliot, "Little Gidding" (the last of his Four Quartets)

In this thesis we moved from the simple algorithmic implementations of ready-made linguistic approaches to Binding Theory towards an original integration of insights drawn from three among its most influential interpretations. Moreover, in an effort to overcome the limitations of each of them, we devised a computational system which generates semantic representations of a sentence which comply with the principles of Binding Theory without running into some computationally intractable problems implicit in the theoretical models. In our approach, linguistic stipulations are subsumed by computational and architectural principles. We prove how our approach effectively accounts for phenomena that asked for somewhat odd stipulations in the theoretical framework.

At the methodological level, algorithmic elegance proved to be a useful guideline to highlight strengths and shortcomings of different approaches to Binding Theory. The algorithmic standpoint provides another criterion, in addition to empirical coverage and cognitive plausibility, to evaluate different models issued from linguistics. The fact that this criterion led us towards more and more refined linguistic interpretations of the principles of Binding Theory, provides in our opinion an *a posteriori* validation of the real progress achieved in this domain. We believe that the path sketched in chapter 4, from off-the-shelf linguistic models towards original computational systems, is an example of the fruitful two-way interaction between linguistics and computer science that should be the main drive fueling the endeavor of computational linguistics.

Although computational efficiency was not the main focus of this thesis (Ristad's [56] results on complexity issues for anaphora resolution presented in section 3.6.6 seem to cast a shadow over any future possible breakthrough on this matter), the endeavor to eliminate the more blatantly NP-complete component from our

system proved to be fruitful. The result is a computational system which derives Reinhart's [50] division of labor between semantic binding and coreference, the latter being delegated to some extragrammatical criterion, which is not devoid of sense from a cognitive point of view. Such vision of the general architecture of language faculty emerges quite naturally, we believe, from the computational assumptions over which our approach is founded, instead than from stipulations exclusively made to cope with empirical data.

If it is true that "there is nothing more practical than a good theory" (Vladimir Vapnik)[1], by taking seriously the major insights of 30 years of linguistic enquiry on Binding Theory and unifying them in a coherent computational framework, we expect to achieve significant improvement in terms of empirical coverage over systems that only rely upon heuristic algorithms. A computational implementation of our algorithm is in order in the immediate future, possibly integrated with John Hale's minimalist parser (see [24], [1]).

Schlenker's somewhat revolutionary paradigm provides another source of inspiration for further developments of our approach. As highlighted in chapter 5, his idea of a memory register over which postulate an economy principle is both intuitive and powerful. He actually achieves a perfectly homogeneous treatment of principles B and C effects, while our approach display a somewhat disturbing redundancy (see the observation at the end of section 4.5.5). This might be taken as a clue that the idea of memory register might be integrated in our approach. At the same time our approach is immune from the problems his runs into with respect to quantification.

A further development is the integration of the present computational framework in the larger context of discourse-related theories like DRT. Although DRT is neutral with respect to the computational mechanisms that account for pronoun resolution, it certainly raises some questions that lie within the scope of our inquiry. In our treatment of intrasentential anaphora we assumed that the output of our computation are first order logic formulas. However, we have seen in section 3.7 that this language seems to fall short even in front of seemingly innocuous sentences like *every farmer who owns a donkey beats it*, and our approach can't but share its flaws. DRT provides a new language to account for the correct semantics of such sentences. We leave a thorough study of the interaction between our computational approach to intrasentential anaphora and the language of DRS as a promising line of future development.

Another important issue that deserves further exploration is the interaction between our computational treatment of anaphora and ellipsis phenomena, like that displayed in the sentence *John loves his mother and Bill does too*. Contrary to Schlenker's, our approach extensively generates all the possible bound-variable readings for a sentence involving personal pronouns. Therefore, as pointed out by Claire Gardent (private communication) it seems well-suited to be integrated with higher-order unification approaches to ellipsis like in [16].

[1] but "this is true much more in theory than in practice" (Christian Retoré, private communication)

References

1. Maxime Amblard, *Réepresentations sémantiques pour les grammaires minimalistes*, Research report 5360, INRIA, 2004.
2. Sergio Baauw and Denis Delfitto, *New views on reflexivity: Delay effects in acquisition, cross-modular principle b and reflexive clitics in romance*, Probus **2** (2005), no. 17, 145–184.
3. H.P. Barendregt, *The lambda calculus (its syntax and semantics)*, North-Holland, 1984.
4. John Barwise and John Perry, *Situations and attitudes*, MIT Press, Cambridge, MA, 1983.
5. Patrick Blackburn and Johan Bos, *Representation and inference for natural language : A first course in computational semantics*, CSLI, 2005.
6. Sean Boisen, Yen-Lu Chow, Andrew Haus, Robert Ingria, Salim Roukas, and David Stallard, *The BBN spoken language system*, Proceedings of a workshop on Speech and natural language (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1989, pp. 106–111.
7. Daniel Büring, *Binding theory*, Cambridge Textbooks in Linguistics, Cambridge University Press, Cambridge, 2005.
8. Gennaro Chierchia and Sally McConnell-Ginet, *Meaning and grammar: An introduction to semantics*, MIT Press, Cambridge, MA, 2000.
9. Noam Chomsky, *Three models for the description of language*, IRI Transactions on Information Theory **2** (1956), no. 3, 113–124.
10. _____, *Essays on form and interpretation*, Elsevier North-Holland, 1977.
11. _____, *On binding*, Linguistic Enquiry **1** (1980), no. 11, 1–46.
12. _____, *Lectures on government and binding*, Foris, Dordrecht, 1981.
13. _____, *Barriers*, MIT Press, Cambridge, MA, 1986.
14. _____, *The minimalist program*, MIT Press, Cambridge, MA, 1995.
15. Noam Chomsky and George A. Miller, *Finite-state languages*, Information and Control (1958), no. 1, 91–112.
16. Mary Darlymple, Stuart Shieber, and Fernando Pereira, *Ellipsis and higher-order unification*, Linguistics and Philosophy **14** (1991), no. 4, 339–452.
17. Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu, *Constraints over lambda-structures in semantic underspecification*, Proceedings of the Thirty-Sixth Annual Meeting of the ACL (San Francisco, California) (Christian Boitet and Pete Whitelock, eds.), Morgan Kaufmann Publishers, 1998, pp. 353–359.
18. Sandiway Fong, *Free indexation: combinatorial analysis and a compositional algorithm*, Proceedings of the 28th conference on Association for Computational Linguistics (Morristown, NJ, USA), Association for Computational Linguistics, 1990, pp. 105–110.

19. Danny Fox, *Reconstruction, binding theory and the interpretation of chains*, Linguistic Inquiry **30** (1999), no. 2, 157–198.
20. ———, *Economy and semantic interpretation*, MIT Press, Cambridge, MA, 2000.
21. L. T. F. Gamut, *Logic, language, and meaning*, vol. 2: Intensional Logic and Logical Grammar, The University of Chicago Press, Dordrecht, 1991.
22. A. Giorgi, F. Pianesi, and G. Satta, *A computational approach to binding theory*, 1990.
23. Jean-Yves Girard, Yves Lafont, and Paul Taylor, *Proofs and types*, ch. 6, Cambridge University Press, 1989.
24. John Hale, *The information-processing difficulty of incremental parsing*, Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together (M. Crocker F. Keller, S. Clark and M. Steedman, eds.), 2004, pp. 58–65.
25. Irene Heim, *Anaphora and semantic interpretation: A reinterpretation of reinhart's approach*, The Interpretive Tract (Percus and Sauerland, eds.), MIT Working Papers in Linguistics, MIT Press, 1993.
26. Irene Heim and Angelika Kratzer, *Semantics in generative grammar*, Blackwell, Oxford, 1998.
27. Jerry Hobbs, *Resolving pronoun references*, Lingua (1978), no. 44, 311–338.
28. ———, *Resolving pronoun references*, (1986), 339–352.
29. Robert J. P. Ingria and David Stallard, *A computational mechanism for pronominal reference*, Proceedings of the 27th conference on Association for Computational Linguistics (Morristown, NJ, USA), Association for Computational Linguistics, 1989, pp. 262–271.
30. Ray Jackendoff, *Semantic interpretation in generative grammar*, MIT Press, Cambridge, MA, 1972.
31. Aravind Joshi and Yves Shabes, *Tree-adjoining grammars*, Handbook of Formal Languages (Rozenberg and Salomaa, eds.), Springer-Verlag, Berlin, 1997.
32. Daniel Jurafsky and James Martin, *Speech and language processing*, Prentice Hall, 2000.
33. H. Kamp and U. Reyle, *From discourse to logic*, D. Reidel, Dordrecht, 1993.
34. Andrew Kehler, *A discourse copying algorithm for ellipsis and anaphora resolution*, Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics (Morristown, NJ, USA), Association for Computational Linguistics, 1993, pp. 203–212.
35. Ewan Klein and Ivan Sag, *Type-driven translation*, Linguistics and Philosophy (1985), no. 8, 163–201.
36. Shalom Lappin and Herbert J. Leass, *An algorithm for pronominal anaphora resolution*, Computational Linguistics **20** (1994), no. 4, 535–561.
37. Howard Lasnik, *Remarks on coreference*, Linguistic Analysis **1** (1976), no. 2, 1–22.
38. ———, *Treatment of disjoint reference*, Journal of Linguistic Research **1** (1981), no. 4, 48–58.
39. ———, *Essays on anaphora*, Kluwer, 1989.
40. Alain Lecomte and Christian Retoré, *Extending Lambek grammars: a logical account of minimalist grammars*, 39th Annual Meeting of the Association for Linguistic Computing, ACL (Toulouse), July 2002.
41. Christopher D. Manning and Hinrich Shütze, *Foundations of statistical natural language processing*, MIT Press, Cambridge, MA, 1999.
42. Roberto May, *The grammar of quantification*, Ph.D. thesis, MIT, Cambridge, MA, 1977.
43. Richard Montague, *The proper treatment of quantification in ordinary english*, Formal Philosophy (1973), 247–270.

44. ———, *English as a formal language*, Formal Philosophy: Selected Papers of Richard Montague, Edited and with an Introduction by Richmond H. Thomason (Richmond H. Thomason, ed.), Yale University Press, New Haven, 1974.
45. Michael Moortgat, *Categorial investigations: Logical and linguistic aspects of Lambek calculus*, Foris, Dordrecht, 1988.
46. Christos M. Papadimitriou, *Computational complexity*, Addison-Wesley, Reading, MA, 1994.
47. Barbara H. Partee, Alice ter Meulen, and Robert Wall, *Mathematical methods in linguistics*, Kluwer, Dordrecht, 1990.
48. Fabio Pianesi, *Indexing and referential dependencies within binding theory: a computational framework*, Proceedings of the fifth conference on European chapter of the Association for Computational Linguistics (Morristown, NJ, USA), Association for Computational Linguistics, 1991, pp. 39–44.
49. Alexis Manaster Ramer, *Review of The Language Complexity Game*, Computational Linguistics **21** (1995), no. 1, 124–131.
50. Tanya Reinhart, *Anaphora and semantic interpretation*, University of Chicago Press, 1983.
51. ———, *Coreference and bound anaphora: a restatement of the anaphora questions*, Linguistics and Philosophy (1983), no. 6, 47–88.
52. ———, *Strategies of anaphora resolution*, Interface Strategies (Hans Bennis, M. Everaert, and E. Reuland, eds.), North Holland Amsterdam, 2000, pp. 295–324.
53. Tanya Reinhart and Eric Reuland, *Anaphors and logophors: an argument structure perspective*, Long-distance Anaphora (Jan Koster and Eric Reuland, eds.), Cambridge University Press, 1991, pp. 283–321.
54. Tanya Reinhart and Eric Reuland, *Reflexivity*, Linguistic Inquiry **24** (1993), no. 4, 657–720.
55. Eric Reuland, *Anaphors, logophors and binding*, Syntaxt and Semantics **33** (2001), 343–370.
56. Eric Sven Ristad, *The language complexity game*, MIT Press, Cambridge, MA, 1993.
57. E. G. Ruys, *The scope of indefinites*, Ph.D. thesis, Utrecht University, 1992.
58. Philippe Schlenker, *Towards a semantic reinterpretation of binding theory*, Proceedings of the Amsterdam Colloquim, 2003.
59. ———, *Non-redundancy: Towards a semantic reinterpretation of binding theory*, Natural Language Semantics **1** (2005), no. 13, 1–92.
60. Edward Stabler, *Derivational minimalist*, Logical Aspects of Computational Linguistics (LACL96), vol. 1328 of LNCS/LNAI, Springer-Verlag, 1997.
61. Mark Steedman, *The syntactic process*, MIT Press, Cambridge, MA, 2000.
62. Richmond Thomason (ed.), *The collected papers of Richard Montague*, Yale University Press, 1974.