Glue Semantics as Dominance Constraints

Master's Thesis

Advisors: Dr. Alexander Koller
Prof. Dr. Manfred Pinkal

Étienne Ailloud
M.Sc. Language Science and Technology

Computerlinguistik
Fachrichtung 4.7 Allgemeine Linguistik
Universität des Saarlandes

Saarbrücken, 18/05/2007

Erklärung

Ich erkläre an Eides statt, daß ich die Masterarbeit mit dem Titel *Glue Semantics as Dominance Constraints* selbständig und ohne fremde Hilfe verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und alle den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Saarbrücken, den 18. Mai 2007

Étienne Ailloud

Abstract

The choice of a suitable semantic representation is crucial in many natural language processing applications. Since sentences of a corpus often have a large number of interpretations according to the syntax-semantics interface, it is a sensible choice to avoid the enumeration of all the possible readings by using a concise representation leaving interpretations unspecified, in a first step. Among these **underspecification** approaches, different formalisms coexist.

Dominance Constraints refers to a family of logical languages that describe trees via constraints. It enjoys the visually intuitive instance of dominance graphs, as well as an efficient solving algorithm for a linguistically relevant fragment.

Glue Semantics is a semantic formalism whose meaning construction process is akin to a proof. Underspecification is implicit in Glue Semantics, and is induced by the multiplicity of proofs from given hypotheses.

Translations between underspecification formalisms are important to make the underspecification mechanisms in each more apparent, and to allow each one to benefit from the advantages of the others.

This thesis defines a translation between Glue Semantics and Dominance Constraints, and proves it sound. It thereby reveals the implicit underspecification of semantic representation in Glue Semantics, and implies the existence of an efficient solving algorithm for a fragment of Glue Semantics. It thus fills a gap between the two formalisms.

Contents

Chapter 1

Introduction

1.1 Ambiguity in semantic formalisms

Meaning is important to many applications of natural language processing. Diverse semantic formalisms have been created to represent the meaning of natural languages utterances. But even with an operational formalism, meaning cannot be always uniquely determined: Ambiguities are inherent to natural language.

To illustrate the issue, we consider a particular sentence:

(1.1) Every woman loves a man.

There are two different readings of this notoriously ambiguous sentence.[1] We state both of them using a standard first-order logical form:

(1.2) (a) $\forall x\ (\mathsf{woman}(x) \rightarrow \exists y\ (\mathsf{man}(y) \wedge \mathsf{love}(x, y)))$ $(\forall\exists)$

(b) $\exists y\ (\mathsf{man}(y) \wedge \forall x\ (\mathsf{woman}(x) \rightarrow \mathsf{love}(x, y)))$ $(\exists\forall)$

This particular sentence may be an easy case for semantic theories (the two readings simply corresponds to twofold alternation between the two quantifiers), but real-life sentences do trigger ambiguity in many occasions.

In fact, the number of readings may grow exponentially in the number of ambiguity triggering configurations (the most basic example being quantifiers). A particularly pathological example from the Rondane Treebank has $2.4 \cdot 10^{12}$ "readings", although the sentence (*But that would give us all day by Tuesday*) looks rather harmless. The design of the syntax-semantics interface is there responsible for the numerous syntactic configurations which may trigger ambiguity. In any case there are too many to apprehend (and process) all readings reasonably. Even when less problematic sentences are considered, the space of possible readings remains a major problem for actual systems.

In either case, be the ambiguity genuinely semantic as in example (1.1) or a design idiosyncrasy in a syntax-semantics interface, the multiplicity of readings

[1] At least for semantic theories. The distinct readings of a semantically ambiguous sentence do not necessary appeal to anybody's intuition. Semantic theories actually dictate which readings should be taken into account by a "reasonable" model. It is not always clear how human readers are supposed to accommodate all distinct readings of a "pathological" sentence, for instance *Someone may fool a single person a thousand times, or fool a thousand persons a single time, but nobody may fool a thousand persons a thousand times.*

must be taken in consideration. The classic approach to ambiguity is the enumeration of all readings (**resolution**) *before* meaning construction. This means that this step is somehow accommodated in the syntax-semantics interface (an example thereof is Cooper Storage, cf. [Kel88]), which implies that meaning construction can become exponentially costly.

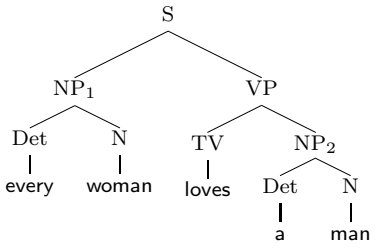
In this case, the construction of a semantic representation required by some natural language system may have drastic consequences on its performances. It is therefore crucial to provide an alternative to avoid this costly enumeration of readings.

Underspecification is a paradigm that addresses this problem.

1.2 Context: Underspecification formalisms

The **underspecification** approach is different from the **resolution** in that it allows not to enumerate all readings in a first step. To achieve this it provides a concise semantic representation of a sentence that is *not yet committed to any reading*. Enumeration of readings can then be performed a posteriori, as a genuine semantic step.

Implementing underspecification. For our previous example, we would like a compact representation for both the $\forall\exists$ and the $\exists\forall$ reading, to be derived from the syntactic parse tree alone, given below:



(1.3)

A closer look at meaning representations for the two readings (1.1) shows that *different parts* of the logical representations contribute to different semantic functions, and correspond in both representations. Though it may not be particularly striking in the "flat" form of the logical formulas (focussing on the two quantified noun phrases, respectively):

(a) $\underline{\forall x\ (\mathsf{woman}(\underline{x})\ \underline{\rightarrow}\ \overline{\exists y\ (\mathsf{man}(\overline{y})\overline{\wedge}\mathsf{love}(\underline{x},\overline{y}))})}$

(b) $\overline{\exists y\ (\mathsf{man}(\overline{y})\overline{\wedge}\ \underline{\forall x\ (\mathsf{woman}(\underline{x})\ \underline{\rightarrow}\ \mathsf{love}(\underline{x},\overline{y}))})}$,

in the canonical tree structures of those formulas (figure 1.1), partial subtrees are found in both readings, but at different places.

Thus, a meaning representation consisting of the different subtrees as meaning chunks, together with a description of how they may be combined would be a good candidate for an underspecified representation. Most underspecification formalisms contain some structures representing meaning elements as well as rules for their possible combinations.

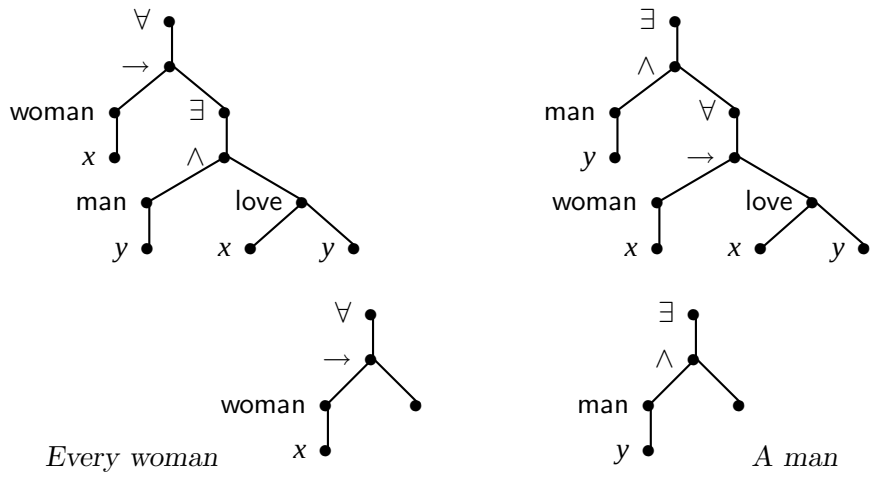


Figure 1.1: Tree representations for the two readings of *Every woman loves a man.* The respective contributions of the two quantifiers are extracted from definite parts of the tree.

This generic principle for underspecification formalisms can be found in a number approaches found in the literature, but we focus in this thesis only two formalisms: **Dominance Constraints** and **Glue Semantics**, which we describe briefly.

Dominance Constraints. The bottom line to the Dominance Constraints approach (see [EKN01]) is that trees can be described by constraints, but also only partially described, which very naturally yields underspecification. Therefore a metalanguage over tree representations has been designed in order to be used within the framework of Constraint Programming.

Dominance Constraints also offer the advantage of having intuitive representations: the dominance graphs. Everything needed to represent meaning is there present in a single drawing: the tree labelling structure, as well as the metalanguage for producing actual representations (viz. trees).

Moreover, the dominance graph version enjoys a very efficient solving algorithm, making dominance graphs the most useful objects for ambiguity resolution.

Glue Semantics. In a nutshell, **Glue Semantics** (see [Dal01]) is a description of how semantic resources are used—consumed and produced. It can be seen as a metalanguage which is borrowed from Proof Theory. In particular, it is based on Linear Logic and the Curry-Howard Isomorphism.

In Glue Semantics, meaning construction is tantamount to performing a proof. The only constraint is thus for the proof to be valid in the chosen proof system. Then ambiguity can be seen as the multiplicity of proofs and its resolution as the choice of a valid proof.

An important aspect, thus, is that contrary to dominance graphs, the underspecification of Glue Semantics representations is only *implicit*: All the material

needed for the proof (viz. the hypotheses) constitutes such a representation. Indeed, we argue in this work that Glue has got (implicit) devices for representing (and solving) ambiguity, which are made explicit using dominance graphs.

1.3 Translations between semantic formalisms

Due to the great variety of different semantic formalisms, the need for translations between them arise frequently. It is therefore important to find reliable translation mechanisms which should have the following properties:

- Soundness, i.e. everything the translation output is a proper representation with proper solutions;
- Completeness, i.e. all expected readings are given by the output (in other words the translation is expressive enough).

Providing a translation mechanism also offers some advantages. Firstly, by revealing, through a translation, the mechanisms dealing with ambiguity in each single formalism, we are able to corroborate the rough idea that the different underspecification formalisms basically share similar means of representing ambiguity, since they separate in a similar way the mechanisms for representing **meaning** parts and those for **assembling** these elements (thus specifying scope, i.e., constraints on how to combine the meaning parts).

In this way, pieces of meaning may be mapped together, while assembly mechanisms can be compared.

Secondly, translations also constitute a possible way of combining advantages from different formalisms. While the dominance constraints approach enjoys intuitive graphs and efficient algorithms, Glue Semantics enjoys the computational insights inherited from the Curry-Howard Correspondence, and offers a very simple and unique composition process (the eponymous "gluing").

1.4 Contribution

The main contribution of our work was to provide a mechanism for translating Glue Semantics representations into dominance graphs. A visual idea of the proof is given in figure 1.2. We show in particular that a similarity can be found between the **dominance** relation between fragments of a dominance graph (top of figure 1.2: dotted edges) and the **conclusion/premiss** opposition (i.e., dependency of pieces of logical material on each other) between Glue axioms (bottom of figure 1.2: root of fragment and assumed subproofs of W and M).

In addition to the translation mechanism, we also give a formal proof of its soundness, i.e. that all of its outputs are dominance constraints that, firstly: are well-formed; secondly: behave as expected, in that they only admit solutions that correspond to solutions of Glue representations.

Additionally, we introduce our translation mechanism in the context of similar works, thus giving a more generic idea of the possible extension of our translation to other paradigms.

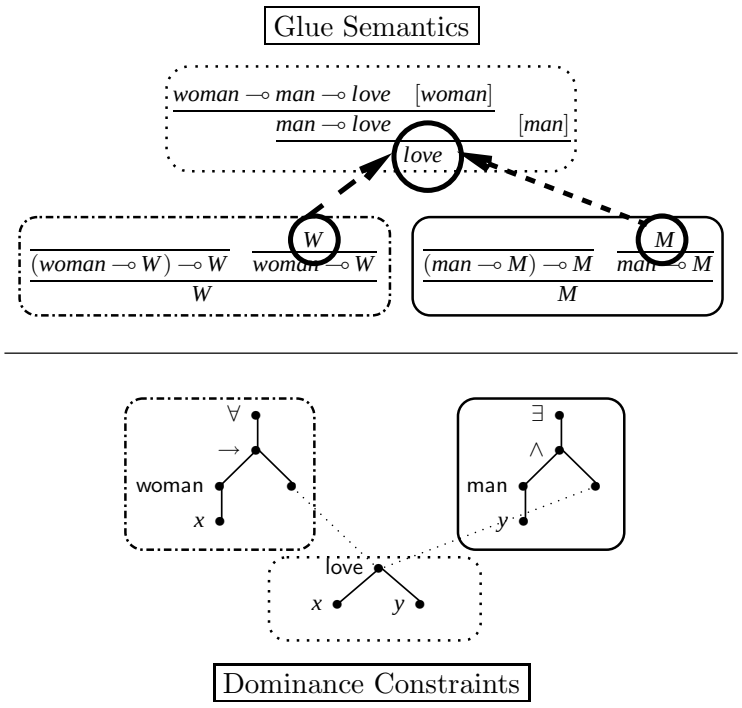


Figure 1.2: A graphical idea of how a translation may work, run on the semantically ambiguous sentence *Every woman loves a man.*
Both M and W may be substituted with *love* and with each other, so that both subproofs may be used in either order.
Likewise, each dominating tree may be plugged on top of the other.

1.5 Outline of the thesis

This thesis is organised in the following manner. Chapter 2 will present Dominance Constraints and its usage in underspecified computational semantics. Chapter 3 will focus on Glue Semantics and its resource-conscious treatment of meaning. Other similar and important works will be briefly presented in chapter 4. The most important part (chapter 5) will expose the work achieved towards the translation aforementioned. Finally, chapter 6 wraps up and gives hints at potential future extensions of the contribution.

Chapter 2

A closer look on Dominance Constraints

2.1 Introduction

Coping with ambiguity. The first formalism we present has its origins in computer science, namely in the Constraint Programming paradigm.

Its application to natural language processing, and more specifically, to underspecified semantics, lies in the following observation.

Traditionally, different readings for a semantically ambiguous sentence yield distinct meaning representations. Let us return to the ambiguous sentence from the introduction:

(2.1) Every woman loves a man.

Its two distinct readings may be represented by the following first-order logical formulas:

1. $\forall x(\mathsf{woman}(x) \rightarrow \exists y(\mathsf{man}(y) \wedge \mathsf{love}(x, y)))$ and
2. $\exists y(\mathsf{man}(y) \wedge \forall x(\mathsf{woman}(x) \rightarrow \mathsf{love}(x, y)))$.

The tree is a more familiar concept in computational linguistics, and by looking at the structures induced by the formulas (figure 2.1), one notices that only disjoint fragments of each tree need to be *rearranged* to switch from a reading to the other. The Constraint Language for Lambda-Structures provides the logical material for such a description.

In fact, one can be more accurate than that: The two fragments corresponding to both quantified noun phrases (henceforth QNP) are in each configuration *above* the fragment modelling the transitive verb. Besides, in one configuration the *every woman* fragment is located above the *a man* fragment (wide scope for *every woman*) and in the other configuration the converse happens (wide scope for *a man*).

This is precisely what the language $\mathcal{DI}$ of Dominance Constraints is able to model. Identifying several parts of trees and *describing* how they may rearrange is the keynote of Dominance Constraints in general.

The dominance ("aboveness") relation in a tree (model-theoretically: in a tree structure) is herein used to interpret the symbol $\lhd^*$ of domination between two

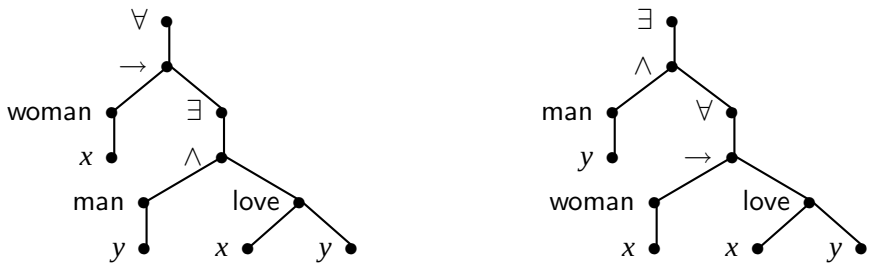


Figure 2.1: Representations for the two readings of *Every woman loves a man.*[1]

constraint nodes, transcribed on the **dominance graph** by the dotted straight lines (figure 2.2): We can check that each of the tree (structure) from figure 2.1 indeed is modelled by the structure from figure 2.2).

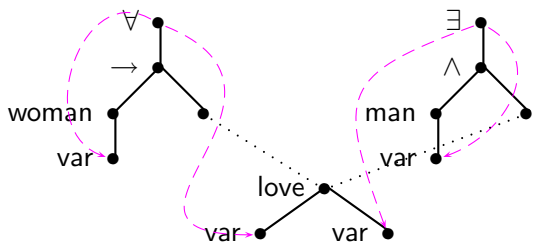


Figure 2.2: An underspecified representation for *Every woman loves a man.*

Furthermore, the drawing from figure 2.2 is a genuinely **underspecified** representation of the ambiguous sentence, because it deploys the maximum (most specific) meaning structure *without committing to one reading in particular*. And in doing so it avoids the (here limited) combinatorial explosion of fully specified syntax-semantics interfaces, mentioned in the introduction (§1.2).

A tiny syntax-semantics interface examplifying the treatment of a few linguistic phenomena is exposed in subsection 2.3.

Resolving ambiguity. Retrieving all possible readings will be achieved by enumerating all **solutions** of the dominance graph. This new concept requires of course a theoretical apparatus (§2.2). The latter is inasmuch important as the generic definition of solution of a dominance constraint is somewhat loose for the

[1] Actually, **binding relations** are used instead of variables: dotted curved edges in figure 2.2. They prove a reliable solution when faced to the usual issues of **variable capture**. They are induced by generic binding specifications: for instance, $\mathcal{DIL}_\lambda$ is the most commonly used for representing meaning in Dominance Constraints ([Kol04, chap. 2]); it corresponds to how λ-calculus accommodates variable binding.

Because binding specifications allow to decouple semantic structure from variable binding, the issue of binding has no influence on the concept of scope resolution. It is thus irrelevant in this thesis, so we may safely ignore it. The constraints developed in the translation (chap. 5) do not make use of constraints for binding either, because the Curry-Howard Correspondence implicitly takes care of it (cf. chapter 3).

sole retrieving of readings for an ambiguous sentence. Any tree that could be *embedded* in the dominance graph and satisfying the proper dominance constraints could be a solution thereof, in particular any tree with more nodes. This leads to an infinite number of actual solutions to any dominance graph representing an ambiguous sentence. Since it is definitely too many for linguistic purposes, another concept is introduced, and one will practically restrict oneself to **constructive solutions** of a given constraint/graph.[2] These solutions are the one that comply best to our intuitions, since they are required to map every of their nodes to every node of the dominance graph in a one-to-one fashion. [3]

Several algorithms have been given for solving dominance constraints, sometimes with very different approaches ([Kol04, chap. 4]). One of the most efficients was given that bases on these tree representations ([Kol04, chap. 5]). Most importantly, it is shown to be tractable for a fragment of input constraints that is conjectured to be representative of "reasonable" syntax-semantics interfaces. Linguistic application is shortly exposed in section 2.3.

A generic formalism. The grounding idea behind Dominance Constraints is that constraints are being used for describing how different graphs may *interact* with each other to form an adequate meaning representation—a tree. But this is all very general.

Firstly, a lot of different interacting relations are likely to be expressed for tree fragments. Secondly, trees may be assigned labels from very different signatures and thus be given several distinct semantics.[4]

Therefore, what we will use and refer to as "Dominance Constraints" in the following (and in particular in our contribution—chapter 5) is but a fragment of a broader, far more general formalism: the Constraint Language for Lambda Structures (CLLS, [EKN01]). In fact, one should rather speak of "*languages* of dominance constraints", because the nature of relations between fragments of trees is manyfold, even for the restricted domain of semantic underspecification.

Different versions of CLLS have been used to model anaphoric relations, for example, or reinterpretation ([Kol04, chap. 7 & 8]). The constraint language used here is the one referred to as $\mathcal{DI}$ in [Kol04], which may express **labelling**, **dominance** and **inequality** of tree nodes. That is, one is able to express:

1. a parent-child relation between nodes, as well as the label borne by the parent;
2. the existence of a path between them, and their relative positions on it;
3. and of course whether nodes are different.

[2] Another way to limit the number of solutions and the one actually used in solving algorithms is to consider so-called **minimal solved forms** of dominance constraints, which are archetypical for any actual solutions that can be embedded in them. The approaches are equivalent, modulo well-formedness conditions: normality and leaf-labeledness (cf. §2.2.3 below).

[3] Some linguistic phenomena may utilise the possibility of having more nodes in solutions as explicitly required. Reinterpretation of knowledge may play this role. They are used in non-trivial extensions of dominance constraints (cf. [KNS00], e.g.).

[4] These two features of the formalism reflect the clear distinction between meaning language and the metalanguage of description, a important distinction followed by all modern underspecification formalisms (such as Glue Semantics, cf. chapter 3).

The signature from which labels are taken may differ according to modelling purposes. Labels canonically used to represent semantic underspecification via Dominance Constraints, as in [Kol04], use the syntactic material of λ-terms and higher-order logic.

The one chosen for our purposes (chapter 5) is able to emulate Linear Logic proof trees, and therefore consists of names of proof rules, with the corresponding arities.

A dual formalism. The easiest way to look at Dominance Constraints is as graphs. Indeed, dominance graphs may also be defined that are proved to be equivalent (at least for scope of the present work) to dominance constraints. That is, there are two ways of seeing and utilising this formalism:

1. The *declarative* version of **dominance constraints**, the logical language;
2. The visual, perhaps more intuitive[5] version of **dominance graphs**, the graph-theoretical counterpart,

both of which are equivalent, *under certain circumstances* ([Kol04, chap. 5]).

These circumstances presuppose some new well-formedness notions, ensuring that the constraints indeed "look like" a tree, and a proper one, too. Such constraints will be called **normal** and are exposed in subsection 2.2.3.

2.2 Definitions

2.2.1 Trees and tree descriptions

Following the distinction between meaning language and metalanguage, we have to define *what* we describe and *how* this description is achieved.

For the "what" part, we will assume labelled trees over a given signature Σ to be the basic objects—trees in the "classical" sense, as familiar to the computational semanticist.

However, the notion of solution of a dominance constraint is a model-theoretical one. So the trees will also be defined as structures over the signature $\Sigma = \{f|^{a(f)}, g|^{a(g)}, \ldots\}$ (often only implicit, though), that specifies which labels decorate the nodes. The arities $a(f), a(g)$ of the labels may also be implicitly given.[6] The set of vertices V_τ of a tree τ is thus always implicitly interpreted as a unary relation.

But we will speak indifferently of trees or the model-theoretic tree structures they induce, which canonically interpret $\lhd^*$, $\neq$, etc, so long as confusion is harmless. We will also need to uniquely identify the nodes of a tree:

Definition 1 (Node address) *Addresses of nodes of a tree (structure) are defined recursively: If n is the root node, it has address ε. Otherwise it is the i-th child of a node with address u, and has then address ui.*

[5] Beyond seeing dominance graphs as more intuitive just because they are graphs, their inner, tree-like structure furthermore corresponds to the inner structure of the well-known and widely-used λ-terms, while their outer structure (the actual dominance edges) provides an insight on the sentence's scopal behavior, this already at a glance.

[6] In the signature chosen in chapter 5, the arities will be those of the modelled deduction rules.

For the "how" part, we will assume some primitive relations on tree nodes, so that every two nodes are in exactly one of those:

- dominance: starting with the **immediate dominance**: $n \lhd m$ if m is a child of n; Canonically induced are the transitive closure thereof: the **dominance** $\lhd^*$ and the irreflexive version, the **strict dominance**: $\lhd^+$. Unless otherwise mentioned, plain dominance ($\lhd^*$) will be conventionally understood;
- **inequality** between nodes ($\neq$);
- **disjointness**: $n \perp m$ if neither of both applies (in which case there is a lowest node that dominates both n and m—the tree being canonically assigned a partially ordered structure with the lowest upper bound property).

And starting from this, we can define a **dominance constraint** over the signature Σ by the following grammar:

$$\begin{array}{lll} C := & A \mid C \wedge C' & \text{(constraints)} \\ A := & X : f(X_1, \ldots, X_{a(f)}) \mid X \lhd^* X' & \text{(atoms)} \end{array}$$

where X, X', X_1,... are **(dominance) variables**, taken among $\mathcal{V}(C)$, the set of all variables appearing in constraint C.

Atoms of the first type are called **labelling atoms**, those of the second type **dominance atoms**.

A key feature of dominance constraints is their ability to be grasped more easily in a visual way: The dominance graph from figure 2.2, for instance, is merely an **encoding** of dominance constraints as they are exposed above:

1. Dominance variables are encoded as tree nodes;
2. Labelling atoms are encoded as **tree edges**, connecting nodes with plain lines, and giving them as outdegrees the atoms' arities;
3. Dominance atoms are encoded as **dominance edges**, the straight dotted lines.[7]

Besides this, there are a number of precautions one has to take to make this encoding proper and ensure one can speak indifferently of dominance *graphs* or *constraints*, of their *nodes* or *variables*: see §2.2.3.

We mentioned in the introduction (§2.1) that the primary aim of Dominance Constraints for semantic modelling is to describe how *fragments* of trees may rearrange and dominate each other to yield different readings. This very important notion of **fragment** (crucial as well for chapter 5) is formally defined here, in both flavours:

- (CONSTRAINT) Any of the reachability components $\{X \in \mathcal{V}(C);\ \exists Y\ (X, Y) \in R_C^{\leftrightarrow}\}$, where $R_C := \{(X, Y);\ X : f(\ldots, Y, \ldots) \in C \text{ for some } f \in \Sigma\}$ denotes the **one-step reachability relation** and $R_C^{\leftrightarrow}$ the reflexive, transitive and symmetric closure thereof;

[7] There would of course be the need of **binding atoms** that correspond to the round dotted binding edges, but we chose to forget about binding altogether.

- (GRAPH) Any connected component of the graph restricted to tree edges.

The following types of variables (or nodes for the graphs) are of great importance in the subsequent chapters, for dominance atoms will be required to go from holes to roots only (cf. compacity, §2.2.3). Visually, they are simply encoded as unlabelled leaves—leaves of the plain-lined trees, of course—and the same roots as trees'.

Definition 2 *A* **hole** *of a dominance constraint is an unlabelled variable, i.e. it does not appear on the left-hand side of any labelling atom. A* **root** *is a variable that does not appear on the right-hand side of any labelling atom.*

The root of a tree-shaped (cf. definition 5) fragment F is unique and will be denoted by $\mathcal{R}(F)$.

2.2.2 Solutions

The following definition model-theoretically specifies what solutions of constraints are: trees over the given signature that can be embedded into the tree parts of the constraints (the labelling atoms) and fulfilling the dominance parts (the dominance atoms). Constructive solutions further require surjectivity of the embedding assignment.

Definition 3 (Solution) *A tree τ is a* **solution** *of the constraint C if there is a variable assignment $\alpha : \mathcal{V}(C) \to V_\tau$ that makes τ's canonical tree structure $\mathcal{M}_\tau$ a model of C: $(\mathcal{M}_\tau, \alpha) \models C$. It is further a* **constructive solution** *if for every node n of τ, there is a variable $X \in \mathcal{V}(C)$ such that $\alpha(X) = n$.*

Starting from this, we will allow ourselves to introduce indifferently trees or their associated tree structures, in particular when dealing with the notion of solution. Even: to speak of trees as solutions of dominance *graphs*.

The constraint view is equivalent to the graph view insofar as:

1. the constraint is always canonically interpreted as given in the encoding above;
2. the constraint is well-formed: this motivates the next subsection.

2.2.3 Well-formedness

Leaf-labeledness is a first well-formedness property. It ensures, as the name suggests, that leaves (viz. variables that do not appear as the head of a labelling atom) are either labelled or lead to dominance edges: leaves in the broader sense (including dominance edges) must be labelled, then.

The equivalence of the graph view (more intuitive) and the constraint view (possibly more expressive) makes use of normality and compactification: Compact normal dominance constraints are equivalent to dominance graphs, in that their notions of solution correspond (cf. below).

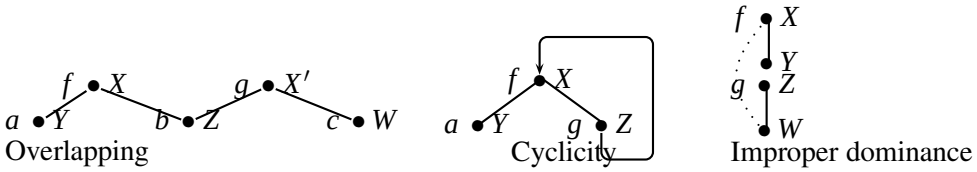


Figure 2.3: Three configurations that are non-normal.

Up to compacity, the properties will be proved fulfilled by the constraints output by the translation from chapter 5.[8]

Normality is defined to rule out the pathological cases illustrated in figure 2.3 which indeed we would not want to be allowed as graphs (the third one being unsuitable for compactification).

Definition 4 (Leaf-labeledness) *A constraint C is leaf-labeled if any of its variables appears on the left-hand side of a labelling or dominance atom.*

Definition 5 (Normality) *A constraint C is normal if it verifies:*

1. (NO OVERLAP) *There is an atom $X \neq Y$ if X and Y are different variables that occur as heads of labelling atoms of C;*
2. (TREE-SHAPED FRAGMENTS) *Every variable appears at most once as a head and at most once as a child in a labelling atom in C, and the one-step reachability relation in C does not allow cycles: its transitive closure R_C^+ is irreflexive;*
3. (DOMINANCES OUT OF HOLES) *If $X \triangleleft^* Y$ is in C, then X is a hole;*
4. (NO EMPTY FRAGMENTS) *Every variable in C occurs in a labelling atom.*

The first point is easily addressed by explicitly completing those constraints that satisfy the first three points with inequality atoms: Constraint $C^{\neq}$ is then defined as

$$C \wedge \bigwedge \{X \neq Y;\ X \text{ and } Y \text{ heads of different labelling atoms}\}.$$

We see in the three pictures of figure 2.3 how normality is lost:

1. The two binary labelling atoms contradict tree-shapedness, as there are atoms $X : f(Y, Z)$ and $X' : f(Z, W)$, so that Z appears twice as a child;
2. There is a cycle via one-step relations: $X : f(Z) \ \wedge \ Z : g(X)$;
3. There is a dominance atom $X \triangleleft^* W$, whereas X is not a hole.

Normal constraints are then proved to be equivalent to dominance *graphs*, insofar as their notions of **solution** coincide: They have the same minimal solved forms ([Kol04, chap. 5]); And minimal solved forms bijectively correspond to constructive solutions of a leaf-labeled constraint ([KNT03]).

[8] Normality and leaf-labeledness of the output constraints of the translation are needed to ensure the notion of **constructive** solutions equivalent to that of minimal solved forms (cf. [KNT03]), which is used by graph algorithms. And normality was necessary for the implementation part (embedding the translation's output into Utool (cf. §2.3.3)), as outputs of arbitrary Glue axioms would have to be displayed as dominance graphs.

Compact dominance constraints are normal dominance constraints that have depth at most 1, in which every variable is either a root or a hole. They are formally defined as the constraints in which every variable occurs in exactly one labelling atom, and where any dominance atom $X \triangleleft^* Y$'s presence implies that Y be a root.

This is not the case of many constraints actually built for computational semantics, and is seldom the case for the new type of constraints developed in this thesis (chap. 5). Nevertheless, there is a straightforward linear time procedure, exposed in [Kol04, chap. 5], that may *compactify* any normal dominance constraint.[9] Hence, it suffices to describe formally the processing of dominance constraints, and their encoding into dominance graphs, for compact constraints.

Compactified normal constraints are fed to the enumeration/solving algorithm, in particular the efficient graph-based solver evoked in §2.3.3 below.

2.3 Utilising Dominance Constraints in computational semantics

We present here a syntax-semantics interface examplifying the linguistic phenomena also covered by the translation developed in this thesis. For this, we assume given a context-free grammar for English, each of its rules triggering a combination of the dominance constraints computed so far. As above, it will be given in the friendlier form of graphs. They uniquely connect to each other via so-called interface nodes, marked by dotted circles in the rules.

As suggested in the introduction (§2.1), dominance graphs (or constraints) provide a compact, underspecified representation for semantically ambiguous sentences. Let us see how it may be constructed (§2.3.2), starting from a tiny syntax-semantics interface (§2.3.1). But the important point remains how to solve dominance graphs and enumerate their solutions, viz. the distinct readings: §2.3.3.

2.3.1 A piece of syntax-semantics interface

Let us assume a context-free toy grammar for English being given, say, sufficient to parse the semantically ambiguous sentence *Every woman loves a man*:

(a1)	S $\rightarrow$ NP VP	(a7')	NP $\rightarrow$ Det N
(a3)	VP $\rightarrow$ TV NP	(a13)	$\alpha \rightarrow W$ if $(W, \alpha, C) \in \mathit{Lex}$

The four items of the syntax-semantics interface from figure 2.4 (taken from a broader one in [Kol04, chap. 3]) correspond each to a grammar rule:

(a1) Expanding the sentence to a verbal phrase and a subject nominal phrase;

(a3) Expanding the verbal phrase to a transitive verb;

(a7') Expanding the noun phrase to a determiner and a noun; linking these together with the adequate dominance.

[9] It is applied by default for constraints output by all distinct codecs of Utool (cf. §2.3.3). Although it will not be exposed in details here, this linear time algorithm proceeds by "collapsing" each fragment into its root and holes—if any, and (since the tree structure is then irrelevant) by introducing a new global variable standing for the tree structure, then finally by raising the dominance edges up to those new roots.

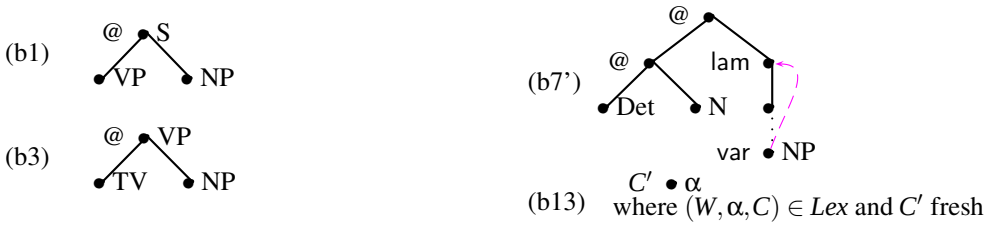


Figure 2.4: A tiny syntax-semantics interface for Dominance Constraints.
'@' denotes λ-application, 'lam' λ-abstraction over the variable occurring at 'var'.

As for the lexicon, *Lex* contains entries of the form: (W, α, C), where W is the word, α its lexical category and C the constraint entry. The single node's constraint C' in (b13) is then constraint C where all variables have been replaced with fresh ones, so as to avoid capturing issues. For instance, proper nouns are raised and require a non-trivial constraint entry (cf. Glue treatment in §3.6).

In each fragment, the **interface node**, viz. the node through which the single fragment may communicate with (i.e. plug into) other ones, is the one noted with the corresponding rule's head. That is, for those four items: S, VP, NP, and α (any category).

It is to note here that the interface node for the quantifier fragment is not its root, but the dominated node standing *below* the actual tree, nevertheless coreferential with its λ-node (hence the binding edge). This particularity, as well as the nodal separation between noun phrase and noun[10], is also to be noted in Glue Semantics (chapter 3). In the end, *the determiner* triggers the ambiguity about what meaning its noun takes, not the noun itself! so it seems normal the determiner should dominate—semantically. But as for the syntax, the NP node is definitely dominating its noun N.

A broader, more realistic syntax-semantics interface, such as that from [Kol04, chap. 3], would of course account for many more linguistic phenomena. In particular, apart from **quantifiers**, other phenomena likely to trigger scope ambiguity, such as: **prepositional phrases** (cf. *Every researcher* of a company *saw most samples.*), **sentence-embedding verbs** (cf. *Every man believes* that a student yawns.), control and raising verbs, or relative phrases (cf. *a book that every student owns*). The first two will actually be rendered by our translation (chapter 5).

2.3.2 Construction: a closer look at (scopal) ambiguity

We already saw in figure 2.2 how the underspecified representation of *Every woman loves a man* looks like; let us sketch now how we get there, starting from the construction rules illustrated in figure 2.4 and the syntactic parse tree:

[10] For the sake of simplicity, the N was introduced in place of a more general $\bar{\text{N}}$, which would possibly trigger further dominance edges, say, via an ambiguous adjectival phrase. Such dominances triggered by the substantive would have nothing to do with that explicitly stated here—intrinsic to the determiner; therefore their respective interface nodes should not be confounded. In Glue, this distinction is instantiated by that between the *var* and the NP resources. In the translation (chapter 5), these resources are mapped to the same interfaces nodes as here.

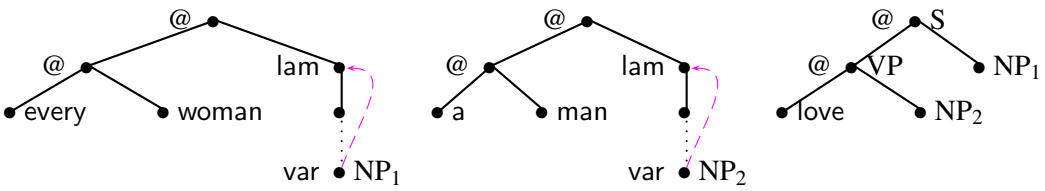
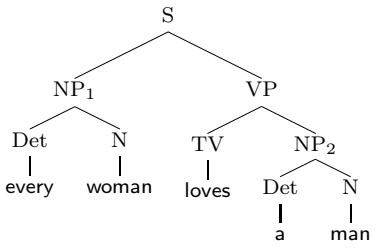


Figure 2.5: Construction of dominance constraints: fragments for, respectively: *Every woman*, *A man* and *loves*.



(2.2)

We may start by plugging the quantified noun phrases (QNP) together (b7+b13, twice), as well as the verbal components (b1+b3):[11] see figure 2.5.

In doing so, we notice that the only remaining task would be to plug the NPs, and thereby make explicit the dominance relations between the meaning components (trees), which were only implicit in (b7'): Both QNP fragments shall dominate the verbal skeleton fragment, yet *at different nodes*, NP_1 and NP_2.

This last difference is easily normalised (cf. §2.3.2 below) in raising both dominance edges to the root S.

Eventually, the form obtained is very much that of figure 2.2, the λ-bindings having already been seamlessly updated via binding edges.

2.3.3 Solving Dominance Constraints

An intuitive solving procedure

Looking back at the dominance graph from figure 2.2, repeated here left of figure 2.6, one can but notice that its incapacity to be a tree lies in the node at the junction of the two dominance edges (node/variable Y on top of the *love* fragment).

Indeed, it is precisely at this location that one is supposed to make a *choice* whether the left fragment shall end up above the right fragment (wide scope for *a woman*) or the other way around (wide scope for *a man*).

And this is also just the way every algorithm step begins, by a **Choice Rule**, illustrated in figure 2.6, that causes the remaining solving to fork in two threads (thus reflecting the potential exponentiality of reading enumeration). Therefrom

[11] In the Glue literature, QNPs are often grouped together as well, prior to any actual derivation. Indeed, unless ambiguity is present in the noun phrase (e.g., in an adjectival phrase), there is no harm in taking these as one-block axioms.

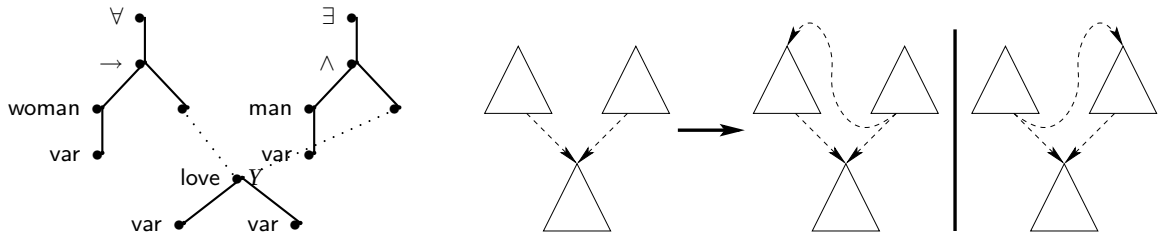


Figure 2.6: The Choice Rule.

results a redundant dominance edge: that which directly connects the higher dominating fragment to the bottom fragment. It is redundant because the domination relation is transitive: The existence of a path from the higher dominating fragment down to the bottom fragment *via* the lower dominating fragment implies a top-down path altogether.

The next step is thus called **Redundancy Elimination**, which gets rid of such dominance edges.

The third and last step consists in ensuring that the edges created (by the choice rule) lead to the highest nodes of fragments, viz. their roots. Since the tree structures of fragments are never changed when solving a dominance graph, it is indeed no harm to move dominance edges to the highest parent of each goal node. This step is called **Parent Normalisation**.

If the resulting graph still contains a node with two incoming dominance edges, the above three-step process is applied again, and so on.

In the end, the only remaining thing is to **check** the resulting graph **for cycles**. If there is a cycle, the input graph is **unsolvable**.

Otherwise, the graph is solvable, and the output is very close to a solution tree: In fact, in most cases (and in all empirical cases, actually), identifying every two nodes connected by a dominance edge will provide a constructive solution tree.[12] And the algorithm enumerates all solution trees through recursive branching at the Choice Rule.

A very efficient solving algorithm

In [Kol04, chap. 5] (improving a result from [KMN00]), another algorithm was exhibited that is able to polynomially enumerate all solved forms of a hypernormally connected dominance graph (cf. chapter 4). The algorithm is quadratic in the size of the graph.

This result alone would motivate the need for translations from diverse underspecification formalisms into Dominance Constraints (chapter 4): Provided the translation were simple enough, the former would benefit from the latter's tractability, in matter of solving and enumerating the solutions.

[12]The algorithm given here actually enumerates *solved forms* of dominance graphs. However, under the Net Hypothesis (cf. chapter 4), which is empirically verified, all solved forms correspond to constructive solution trees in a one-to-one fashion.

Glue Semantics, presented in the next chapter, is a successful example thereof; The proof is exposed in chapter 5.

A word on Utool

This idea is also materialised in a multi-formalism implementation, called Utool[13], that allows to **convert** underspecified semantic representations from and into Dominance Constraints, and of course to solve them with the most efficient algorithms available.

Dominance graphs and solutions thereof are also **displayed**, allowing quick monitoring and experimenting on converted representations. As it happens, part of the translation exposed in chapter 5 was tested on an experimental Glue Semantics codec for Utool.

[13] `http://www.coli.uni-saarland.de/projects/chorus/utool/`

Chapter 3

A closer look on Glue Semantics

3.1 Introduction

Construction as proof. Glue Semantics (or short—Glue)[1] is probably best remembered as "LFG's semantics", but is also a fully-fledged semantic formalism with a genuine, yet implicit, device for realising underspecification. The "glue" part of the name refers to its construction principle, and gives a hint on how semantic composition is performed: meanings for lexical entries are just glued together—following a certain number of rules, of course, borrowed from Linear Logic (henceforth LL). But there is only one uniform principle of composition, that is thus tantamount to performing a proof.

The entries are then linear logic formulas, considered as **axioms**, that is, as hypotheses to the proof to be done. For *Every woman loves a man*, e.g., one would be to prove f from:

$$\begin{array}{l} h \multimap g \multimap f \\ \forall G\ (g \multimap G) \multimap G \\ \forall H\ (h \multimap H) \multimap H. \end{array} \tag{3.1}$$

But admittedly this is not very helpful to our preliminary intuitions, because: what do these symbols represent? what do we want to prove? where does the *meaning* appear?

1. The symbol $\multimap$ is called **linear implication** and is only one of the whole range of new operators Linear Logic provides. Let us understand it as an implication, at first, but more intuitive an implication that we are used to in Propositional Logic: Linear implication actually *consumes* its hypotheses and *produces* its conclusions, ensuring that, roughly speaking, "everything is used and only once" (cf. §3.3.1).

   And this last parsimony concern, or more frequently called in the literature: the **resource-consciousness** of LL proofs, resembles much those known in the LFG world as **completeness** and **coherence**. What hypotheses

[1] A state-of-the-art reference is [Dal01], but the literature is quite important on this formalism; Our second main reference is [Dal99], which deals with some more elaborated aspects of Glue, more experimental ones, too.

and conclusions are to LL proofs, f-structure nodes are to the predicate-argument structures (PRED) they belong to: resources, all of which singly necessitated.[2]

The analogy is not fortuitous: Glue atomic subformulas (lower-case: f, g, h) represent indeed nodes of the sentence's f-structure, while upper-case variables (G, H) may stand for any Glue formula (hence the quantifications).

Upon examining the f-structure for the above sentence:

$$f:\begin{bmatrix}\textsc{pred} & \text{`love'}\langle\textsc{subj},\textsc{obj}\rangle \\ \textsc{subj} & g:\begin{bmatrix}\textsc{pred} & \text{`woman'}\\ \textsc{spec} & \text{`every'}\end{bmatrix} \\ \textsc{obj} & h:\begin{bmatrix}\textsc{pred} & \text{`man'}\\ \textsc{spec} & \text{`a'}\end{bmatrix}\end{bmatrix},$$

the names of atomic subformulas from (3.1) become less cryptic: They correspond to f-structure nodes.[3]

2. Then, proving f turns out to be necessary to get the meaning for the whole sentence, while subproofs will yield subparts of meaning—according to compositionality. An example of a proof of f for the sentence is given below, where one notices that linear implication behaves just like one expects—only that all hypotheses are carefully consumed just once each:

$$\dfrac{\dfrac{\dfrac{}{h\multimap(g\multimap f)}\,(\text{ax}) \quad \dfrac{\dfrac{}{\forall H\ (h\multimap H)\multimap H}\,(\text{ax})}{(h\multimap(g\multimap f))\multimap(g\multimap f)}}{g\multimap f} \quad \dfrac{\dfrac{}{\forall G\ (g\multimap G)\multimap G}\,(\text{ax})}{(g\multimap f)\multimap f}}{f} \tag{3.2}$$

Computational vs. meaning content. As for the meaning, once the proof completed, that does not actually appear in the pure LL representation of (3.1). Anyway, it is sufficient (and in fact necessary) to have proved the LL part to be able to retrieve the global meaning for a sentence, thanks to the **Curry-Howard Correspondence**: The LL formulas used in proving always come along with a meaning part (which is traditionally skipped in Glue derivations for the sake of

[2] An f-structure is **locally complete** if it contains all the governable grammatical functions its PRED governs; It is **locally coherent** if, conversely, all its governable grammatical functions are present in the PRED's argument list. An f-structure is **complete** (resp. **coherent**) iff all its substructures are locally complete (resp. locally coherent). Therefore any node of an f-structure is to be used, and exactly once, just like hypotheses of a LL proof.

For more details on the objects used in Lexical-Functional Grammar, see for instance [DKMZ95].

[3] This notation is abusive, and the example voluntarily very simplified. Besides, the argument structure within the f-structure (PRED) does not necessarily reflect the *semantic* argument structure; more details in §3.2.2.

displayability), so we would actually consider the following derivation:

$$\dfrac{\begin{array}{r}\lambda y.\lambda x.\mathsf{love}(x,y) \;:\; h \multimap g \multimap f,\\ \lambda P.\mathsf{every}(\mathsf{woman},P) \;:\; \forall H\ (h \multimap H) \multimap H,\\ \lambda Q.\mathsf{some}(\mathsf{man},Q) \;:\; \forall G\ (g \multimap G) \multimap G\\ \vdots\end{array}}{\forall x(\mathsf{woman}(x) \rightarrow \exists y(\mathsf{man}(y) \wedge \mathsf{love}(x,y))) \;:\; f} \tag{3.3}$$

The composition[4] process is not particularly complex but is two-faceted, for this is where the Correspondence plays its role:

1. Linear Logic rules dictate how the meaning statements, consisting of a pure LL part and a meaning language part, are combined (metalanguage);
2. The modifications thus undergone by the LL formulas (viz. the LL proof tree) are one-to-one mapped to modifications in the meaning language parts, the root f of the proof tree thus being eventually mapped to the final reading (meaning language).

Section 3.3 will take a closer glance at Linear Logic, in particular at the Correspondence (§3.3.2).

Implicit underspecification. One may notice that, apart from that of (3.2), there is another proof tree that derives f starting from the same hypotheses (3.1), displayed below:

$$\dfrac{\dfrac{}{g \multimap (h \multimap f)}\,(\text{ax}) \quad \dfrac{\dfrac{}{\forall G\ (g \multimap G) \multimap G}\,(\text{ax})}{(g \multimap (h \multimap f)) \multimap (h \multimap f)}}{h \multimap f} \quad \dfrac{\dfrac{}{\forall H\ (h \multimap H) \multimap H}\,(\text{ax})}{(h \multimap f) \multimap f}}{f} \tag{3.4}$$

There the verbal axiom $h \multimap (g \multimap f)$ is replaced by the equivalent formula $g \multimap (h \multimap f)$—the identity is valid in Classical as well as Linear Logic (cf. §3.3 for a justification, §3.4 (3.8) for a proof).

Indeed, multiplicity of proofs is a frequent phenomenon, even when, as Linear Logic strives to, proofs are somehow normalised (the differences between distinct proofs are then *essential*).

As it happens, the λ-term derived from (3.4) corresponds to the expected other reading of *Every woman loves a man*, namely:

$$\exists y(\mathsf{man}(y) \wedge \forall x(\mathsf{woman}(x) \rightarrow \mathsf{love}(x,y))).$$

So there may be a manner of considering underspecified Glue structures, even though only implicitly, namely via the axiom set $\{h \multimap (g \multimap f),\ \forall G\ (g \multimap G) \multimap G,\ \forall H\ (h \multimap H) \multimap H\}$. The formulas' relationships with each other (to be compared with the **dominance** relation in §2.2) are already stated *within* their

[4] *Composition* is the name of meaning *construction* in **compositional** semantic formalisms, as are Dominance Constraints (chapter 2) or Montague Semantics ([Mon74]): "The meaning of the whole is a function of the meanings of the parts."

computational content, thanks to the Curry-Howard Correspondence. The crux is that axioms depend one one another, as a proof may depend on an assumed result (to be proved someplace else). For example, the verb axiom $h \multimap (g \multimap f)$ depends on the *every woman* axiom because it is, modulo substitution of the scope variable $H \mapsto (g \multimap f)$, a *premiss* to $(h \multimap H) \multimap H$. Likewise, it also depends on the *a man* axiom, since it is equivalent to the formula $g \multimap (h \multimap f)$, so that altogether we may see a similarity with figure 2.2, for instance.

Chapter 5 will show that this vision of Glue constitutes in fact a strong link to semantic underspecification, strong enough to support a translation between formalisms.

3.2 "LFG's semantics"

As a part of Lexical-Functional Grammar, originally, Glue Semantics inherited a syntax-semantics interface starting from the f-structures (§3.2.2), although it could be provided for other input structures (cf. §3.2.3).

The f-layer happens to be an intermediate, largely language-independent layer and also contains some higher-level, perhaps even semantic, information. Thus, unlike classic syntax-semantics interfaces which typically map **context-free rules** onto logical formulas, the input of our meaning entries will not be constituted of information about syntactic categories, but already instantiated **f-structure nodes**. This is a peculiarity of Functional Grammar, not of Glue Semantics.

In fact, the σ-layer is intended to play this role of *interface* with classic semantic formalisms (not in the sense of *syntax-semantics* interface, though, for the input is not purely syntactic). For instance, quantifiers are given a **restriction** and a **variable** argument via σ-application, as is needed in Montague-like semantics (higher-order scope variables and first-order variables), say, or in MRS (ordinary variables and handles).[5]

3.2.1 LFG's legacy: Layer structure

Due to the peculiarity of LFG's layer structure (cf. figure 3.1), and more precisely, to its having two separate syntactic layers, the input of the interface will not be a classic context-free grammar. In fact, it will bear no relationship with the surface form of a sentence (this would be a matter for the c-layer), but will be an intermediate representation form—intermediate between syntax and semantics already.

Glue, built upon this layer architecture, makes use of the f-layer and implements the σ-layer, as illustrated in figure 3.1.

The literature on LFG has been quite evasive about concrete specifications of the σ-layer; It merely requires there to be a layer whose nodes (mapped from the f-layer under σ) correspond to "units of semantic structure" ([DKMZ95, page

[5]The f-structures are so much above the surface structure that they can be considered as containing semantic information already. But this is so primitive semantic information that it is in fact underspecified (w.r.t. scope ambiguity). For instance, they were used as input for a direct translation to UDRT in [vGC99].

More classically though, their thematic-like argument structure has been used as the basis to an argument-mapping theory (as in [DLS93], e.g.).

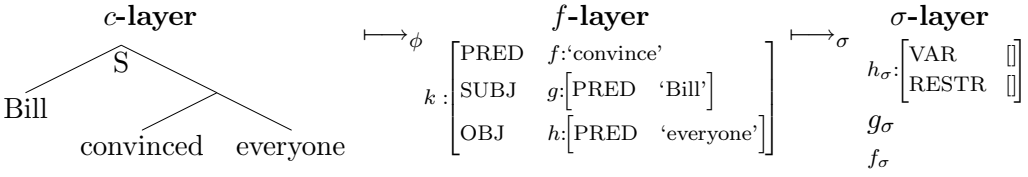


Figure 3.1: LFG's layer structure.

23]), but this is not further specified by LFG. Indeed: this is a matter for the syntax-semantics interface, and not for the lexical-functional grammar itself.

LFG is also a *functional* formalism, and representations are *projected* from one layer to the upper. But within the f-layer itself, every path between nodes (of the Attribute-Value Matrix) may be described as a the successive application of labels from the outer node onto the inner node, yielding *equations* like $(f\ \textsc{obj}\ \textsc{pred}) =$ 'everyone' for instance. Glue's syntax-semantics interface uses such descriptions, thus should also be able to describe such paths.

3.2.2 Instantiation of generic entries

The link between the f-structure evoked in the introduction and the Glue axioms it induces is not very rigorous. Let us consider a smaller example, assuming, for the sake of simplicity, that proper nouns may receive a simple, non-raised treatment, and that *everyone* is interpreted as one block (as usually done in the Glue literature). f-structure, σ-structure and axioms are displayed in table 3.1.

$$f : \begin{bmatrix} \textsc{pred} & \text{'love'}\langle \textsc{subj}, \textsc{obj} \rangle \\ \textsc{subj} & g : \begin{bmatrix} \textsc{pred} & \text{'Bill'} \end{bmatrix} \\ \textsc{obj} & h : \begin{bmatrix} \textsc{pred} & \text{'everyone'} \end{bmatrix} \end{bmatrix} \qquad \begin{array}{l} g_\sigma\ [] \\ h_\sigma \begin{bmatrix} \textsc{var} & v\ [] \\ \textsc{restr} & r\ [] \end{bmatrix} \end{array} \qquad \begin{array}{l} g_\sigma \\ h_\sigma \multimap g_\sigma \multimap f_\sigma \\ \forall H\ (h_\sigma \multimap H) \multimap H. \end{array}$$

Table 3.1: f-structure and interface entries for *Bill convinced everyone*.

The LL atoms actually appearing in the Glue formulas are actually the *images* of the f-structure nodes under σ, not the nodes themselves. It moreover makes sense, in that a "reasonable" semantics should only use the highest available representation layer.

Convention. Yet we will use the same letters for denoting f-structure nodes and their images under σ. Since the topic of this thesis stays within semantics, we will never need to have trace back down the f-structures; It is therefore a safe convention.

But the actual entries of the LFG lexicon still do not look quite as in table 3.1, because the lower-case letters refer to an f-structure, which is still to be constructed from the c-structure. Actual entries are thus *generic* entries (suitable to *any* f-structure), that still have to be *instantiated*. Say, in the case of our example *Bill convinced everyone*:

PN $(\uparrow\ \textsc{pred}) =$ 'Bill'
bill : $\uparrow_\sigma$

TV $(\uparrow \text{ PRED}) = \text{'convince'}$
$\lambda y.\lambda x.\mathsf{convince}(x, y) : (\uparrow \text{ OBJ})_\sigma \multimap ((\uparrow \text{ SUBJ})_\sigma \multimap \uparrow_\sigma)$
NP $(\uparrow \text{ PRED}) = \text{'everyone'}$
$\lambda P.\mathsf{every}(X, \mathsf{person}(X), P(X)) : \forall S\ (\uparrow_\sigma \multimap S) \multimap S$

The first line pertains to the f-structure and sets the needed attribute-value couple; no genuine semantics is involved yet. Because $\uparrow$ is a metavariable (instantiable to any f-node), it could read as "Whenever an f-structure is encountered that through a PRED label leads to a node 'Bill', then the subsidiary f-structure is assigned the meaning Bill (by means of σ-projection).

Now this generic meaning gets *instantiated* to an actual meaning pertaining to the f-structure under consideration, g. With this instantiation step, one leaves the syntax-semantics interface to enter an intermediate level, the σ-layer, committed to one actual structure.

The transitive verb *convince* gets instantiated to the Glue formula $h_\sigma \multimap g_\sigma \multimap f_\sigma$, which means that the binary predicate convince needs to be *discharged* of its two arguments, that happen to be the σ-projections of g and h, respectively.

Linear implication then ensures that each argument is discharged exactly once (cf. §3.3), in accordance with the verb's argument structure (and especially to its semantics, as we will see in §3.4).

Convention. As a second convention, we will always work with already instantiated entries (this amounts to supposing the f-structure always given).

Remark. Although Glue inherits the layer philosophy from LFG, it does not take over **functional uncertainty** (cf. [DKIZ95]): the semantic entries must be deterministically instantiated in order for Glue to function correctly. Equations with multiple paths within an f-structure, as often necessary to syntactic phenomena, are not wanted here.

3.2.3 The two facets of the lexicon

Different notations There have been several notations for the Glue formalism since its beginnings. Consider the example of a lexical entry for the verb *love*, given in the "thematic" (cf. [DLS93]), the old (cf. [vGC99]) and the newer notation (cf. [Dal01], from 1999 onward):

Thematic Glue $\forall x.\forall y.[\mathbf{agent}](g_\sigma, x) \otimes [\mathbf{patient}](g_\sigma, y) \multimap f_\sigma = \mathsf{love}(x, y)$
Old Glue $\forall x \forall y.(r \rightsquigarrow x \otimes s \rightsquigarrow y) \multimap t \rightsquigarrow \mathsf{love}(x, y)$
New Glue $\lambda x.\lambda y.\mathsf{love}(x, y) : s \multimap (r \multimap t)$

The second line could read like: "If two σ-nodes r and s bear, respectively, meanings x and y, then another node t bears meaning $\mathsf{love}(x, y)$." The third line, in a more functional fashion, would say: "The function love of two individuals bears type $r \multimap (s \multimap t)$", which, in a **typed λ-calculus**, means exactly the same.

The newer notation is partly motivated by the Curry-Howard Isomorphism; It really *separates* meaning ($\lambda x.\lambda y.\mathsf{love}(x, y)$) and glue ($s \multimap (r \multimap t)$) language in the statements, whereas the older one kept them just a little mixed. As a side effect, the $\otimes$ connective need not be introduced yet in the newer one—all the better for uniformity.

Arguably, the original is better for *understanding the premisses* (it is more or less enough to read them literally to apprehend them), while the newer is good for *understanding the derivation* (in a proof-theoretical or computational way, cf. §3.3.2 below). Therefore, only the newer will be used in the following.

Meaning language vs. gluing language. We notice therein a clear-cut distinction between **meaning language** and **metalanguage**, as has been exhibited for Dominance Constraints, too (§2.2), and which is arguably required for a satisfying treatment of underspecification in computational semantics.

Again, this feature allows for a relative *liberty in the choice of the meaning language*, although higher-order, Montague-like logics seem a canonical choice. Indeed, Glue Semantics, or in a broader sense: **resource-conscious logics**, have been applied to various forms of meaning representation, such as: Discourse Representation Theory (cf. [vGC99]), Head-driven Phrase Structure Grammar (cf. [AC02b]), Categorial Semantics (cf. [DGLS99]). All of these may thus be processed in a uniform way through a Linear Logic **assembly language**.

3.3 Linear Logic

Linear Logic is a branch of Proof Theory initiated in 1987 by Girard ([Gir87, Gir95]) and distinguishes itself from other proof-theoretical approaches in that it centers the notion of **resource** in a proof: the **logical material** used therein, instantiated by our intuitive notions of axioms, hypotheses or logical consequences, is given a more "natural" interpretation.

The basic idea was that proofs, as they are usually represented (typically, using a tree representing logical deductions) do not reflect accurately the actual process of proving. Usage of resources is ill-described in the classical deduction toolkit, that only says what one is *allowed* to deduce, but not what one is *likely* to deduce. So Linear Logic aims at describing proofs as one would really perform them.

It is also, thanks to the Curry-Howard Correspondence (§3.3.2), a great descriptor of functional programming languages, which proved extremely useful in Computer Science, but also in Computational Semantics in the framework of this thesis. Since the works of Montague ([Mon74]), natural language semantics seem indeed indissociable from λ-calculus, or variants thereof.

3.3.1 A different paradigm

Filling truth tables. A thing that has long been criticized by many logicians (and thus a motivation for the introduction of several non-classical logics) is the unnaturalness of our classic, model-theoretical implication, with truth table as follows:

φ	ψ	$\varphi \rightarrow \psi$
** 0	0	1
* 0	1	1
1	0	0
1	1	1

The first two lines, while fully accepted among mathematicians, specify that an implication be **vacuously true**, i.e. even with unfulfilled premiss. The first

one may even be regarded as less satisfactory (depending on one's intuition or philosophical obedience), since it defines a genuinely empty tautology.

In fact, even though mathematicians use this notion of implication as a tool provided by their logician colleagues[6], it is much likely that their own *intuitive* notion of implication expects another behavior, too, even for the other rows of the truth table: That a simple tautology as "the square of an even number is even" implies Fermat's theorem, no logician would doubt (both assertions being proved to be true in the same axiomatic system), but from an *empirically* mathematical point of view, this would rather sound highly dubious...

It is the very notion of **truth table** that causes the problem: In order to define implication model-theoretically, those lines *had* to be filled. In applications of Logics to Computer Science, this matter became all the more important, for computers are not as accommodating as humans and must be told what to do *in all cases*.

Redefining logic. In Linear Logic, the hypothesis of an implication has to be actually used in order to make it valid, and the syllogical sequent $\varphi, \varphi \multimap \psi \vdash \psi$ as well; but also, once used, it cannot be reused anymore.

It is to this extent a **linear**—as opposed to **monotonic**—logic, in that it has the particularity of *consuming* its premisses upon *producing* its conclusions, in much the same way as a chemical reaction consumes the compounds available in certain proportions and produces its final products in fixed proportions as well.

The following comparison, taken from [Gir95], considers the basic reaction producing water from dihydrogen and dioxygen:

$$2H_2 + O_2 \longrightarrow 2H_2O$$

Every two molecules of dihydrogen combine with one molecule of dioxygen to form two molecules of water. Loosely using the linear logic conjunction and implication, this could be described as:

$$H_2 \otimes H_2 \otimes O_2 \multimap H_2O \otimes H_2O,$$

clearly establishing the *proportions* in which the elements rearrange. Observing combining and production of chemical compounds under this light gives a first idea of how linear conjunction and **linear implication** behave.

In particular, the otherwise widely accepted principles of **strengthening of hypotheses** and **weakening of conclusions** are no longer sustainable in Linear Logic. Compare:

$$A, A, (A \to B) \vdash_{FO} B \quad \text{vs.} \quad A, A, (A \multimap B) \not\vdash_{LL} B \quad (\text{but} \vdash_{LL} A \otimes B).$$

[6] In fact, before the Hilbert programme and earlier attempts to formalise the foundations of mathematics (e.g. Whitehead and Russell's *Principiae Mathematicae*) at the beginning of the 20th century, mathematicians had little care about foundational issues of their discipline. The monstrous sets resulting from Cantor's attempts to found a mathematical theory at the end of the 19th century were only to discourage them (cf. Cantor's paradox). Gödel, later, proved it to be an ultimately hazardous issue. With such antecedents, it is no wonder why mathematics and logics have remained rather separate fields for so much time, in particular why the implication remained so long unexplored. Some recent research topics, though (modern Set Theory, Non-Standard Analysis, Girard's Linear Logic, e.g.), tend to gather them again; but above all the enormous development of Computer Science since more than half a century gave rise to new research in Logic.

Not only does linear implication consume its input, it also *produces* its output, in the same resource-caring way as for hypotheses, that is, it consumes no more and no less than the proof allows. This again bans the principle of *weakening of conclusions*:

$$A, A \to B, C \vdash_{FO} B \quad \text{vs.} \quad A, (A \multimap B), C \;\; \nvdash_{LL} \;\; B \qquad (\text{but} \;\; \vdash_{LL} \;\; B \otimes C).$$

3.3.2 The Curry-Howard Isomorphism

Proof Theory as a major part of (Mathematical) Logic on one hand, and the semantics of programming languages on the other hand, describe both the same!

This observation has become one of the most important paradigms in computation nowadays: the so-called **Curry-Howard Isomorphism** (or **Correspondence**, abbreviated in CHI in the following). One is able to describe **proofs as programs** or, conversely, **formulas as types** (both expressions are other casual denominations induced by the Correspondence).

Any term in a **typed λ-calculus** is assigned a **type** which is a formula from a given logic.[7] This formula is valid if the term is well-typed. Conversely, the term itself represents a proof that its type is valid, via the Correspondence.

In Computer Science, where programs are often modelled as λ-terms, their specification (type) is valid iff their computational content (λ-term) encodes a valid proof thereof.

A double-sided notation. Let us now show how the CHI translates to for the classic implication in Natural Deduction logic, where it is described via two rules. Introduction thus very naturally corresponds to λ-abstraction while elimination corresponds to β-reduction:

$$\frac{t : A \to B \quad u : A}{t(u) : B}\ (\to)_{\text{elim}} \qquad \frac{\begin{array}{c} x : [A]^i \\ \vdots \\ t : B \end{array}}{\lambda x.t(x) : A \to B}\ (\to)_{\text{intro}}$$

This illustrates the proofs-as-programs version of the Correspondence: We have a functional reading of the **modus ponens** (elimination), in that an implication of B from A is also the type of a function $A \to B$, so every image under it should have type B. Likewise, making an **hypothesis** of type A in a derivation leading to B hints at a functional construct, of type $A \to B$.

Linear Logic provides a strictly identical scheme for its linear implication, as shown in table 3.2 below. (Notice herein the perfect adequacy of linear implication to the CHI; one feels indeed that this notion of implication "must be right".)

[7] Usual examples herefrom are Intuitionistic Propositional Logic (simply typed λ-calculus), Girard's F and AF_2 systems; the richer the logic, the more expressive the formalism on the other side. For a long time it was thought that only constructive logics fitted in this scheme, until Griffin ([Gri90]) and Krivine ([Kri03]) proposed λ-calculi with devices "implementing", respectively, the Double Negation elimination (or equivalently, the law of the Excluded Middle) and the Choice Axiom, thus closing the case: All modern mathematics, that is, those based on the Zermelo-Fraenkel axioms *plus Axiom of Choice*, can now be modelled by a CHI-like correspondence.

This has motivated the newer notation for Glue, which separates the computational part (LL formulas) from the meaning part (theoretically whatever, but canonically λ-calculus), as in the entry for a transitive verb (σ-structure only):

$$[\textbf{convinced}] := \lambda y.\lambda x.\mathsf{convince}(x, y) \ : \ h_\sigma \multimap g_\sigma \multimap f_\sigma \tag{3.5}$$

Of particular importance for Glue Semantics is here the *bidirectional constraining* of each of the two sides onto the other one: With every gluing step (cf. next section) performed—right-hand side, the meaning part—left-hand side—gets updated. Therefore, the meaning part is only needed in the lexicon; all proofs may be performed as a pure LL part (which often saves a great deal of place), the final reading being always available upon rereading the proof tree: the CHI is used as a kind of dictionary.

If another meaning language is preferred to λ-calculus (cf. §3.2.3), it should accommodate the LL rules in its own fashion (on the left), but still be "CHI-compliant" (on the right): In particular, it should have an equivalent to the couple **λ-abstraction/application**, which constitute really a powerful combination.

3.4 The semantics of gluing: Meaning construction

With lexicon entries such as (3.5) available, everything is ready for meaning construction. It takes the form of a proof, whose premisses are the CHI-like statements. Their left-hand sides compositionally update the meaning, but the proof itself is constrained by the sole right-hand sides to be led properly. The premisses undergo a single uniform construction process, the eponymous gluing.

This reveals a key feature of Glue: that virtually all the needed linguistic information is contained already in the instantiated lexical entries (implicit in the computational content of the CHI). This way, a simple principle (gluing) can handle complex composition. As such Glue departs from other formalisms, where much more information lies in the assembly itself.[8]

Conjugating premisses. Considering all the statements amounts to setting their *conjunction* as the proof's hypothesis. However, since it is a conjunction-free LL fragment we consider, we will most simply see this in a sequent fashion, say, for *Bill yawned*:

$$[\textbf{bill}], [\textbf{yawned}] \vdash [\textbf{bill-yawned}],$$

abbreviated from:

$$\lambda x.\mathsf{yawn}(x) \ : \ b \multimap f, \ \mathsf{bill} \ : \ b \ \vdash \ \mathsf{yawn}(\mathsf{bill}) \ : \ f$$

and referring to the f-structure:

$$f \begin{bmatrix} \textsc{pred} & \text{'yawn'}\langle \textsc{subj} \rangle \\ \textsc{subj} & b \begin{bmatrix} \textsc{pred} & \text{'Bill'} \end{bmatrix} \end{bmatrix}.$$

[8] For instance, MRS has got two distinct composition principles, for scopal and intersective combination ([CFS01]).

The proof in this case is straightforward, and the derivation tree thus very simple:

$$\dfrac{\dfrac{\lambda x.\mathsf{yawn}(x) \ : \ b \multimap f \quad \mathsf{bill} \ : \ b}{(\lambda x.\mathsf{yawn}(x))(\mathsf{bill}) \ : \ f}\ (\multimap_e)}{\mathsf{yawn}(\mathsf{bill}) \ : \ f}\ (\beta\text{-reduction}).$$

One notices here that implication elimination alone does not exactly lead to the expected result, but to a β-equivalent longer term, which is of no concern to the LL part. It makes no difference for the meaning either, β-reduced terms are just more readable.

Convention. Therefore, in forthcoming derivations, we will assume **β-reductions** to be performed whenever possible. This will allow us to just skip them, so that they will in fact not appear at all.

Sortal restrictions. The space of possible derivations is also limited by other factors than sheer LL validity. Similar sortal restrictions as in type-theoretical semantics are applied, so that transitive verbs, for example, are required to have type $\langle e, \langle e, t\rangle\rangle$ (consume two **individual** arguments to produce a **proposition** result).

For Glue, however, such restrictions apply probably only *for the instantiation step*, in order to determine onto which f-nodes the entries' metavariables $\uparrow$ are mapped. In particular—and this cannot be determined at instantiation step yet, the scope variables may not be of the individual type e.

$$\forall S^{(t)}\ (\uparrow_\sigma^{(e)} \multimap S) \multimap S \qquad \overset{\text{inst.}}{\longmapsto} \qquad \forall H\ (h^{(e)} \multimap H^{(t)}) \multimap H^{(t)}$$

They may thus further constrain the Glue premisses to combine the way wanted and encode, say, scope constraints additionally prescribed by some linguistic theory (cf. [Dal99, pp 47–52]). Yet once the generic entries are anchored to an actual sentence (via the f-structure), little freedom remains, really, as to combination of premisses.

From the instantiated premisses, one sees that H could only be instantiated to f:

$$\begin{array}{ll} [\textbf{bill}] & \mathsf{Bil} : g^{(e)} \\ [\textbf{convinced}] & \lambda x.\lambda y.\mathsf{convince}(x,y) : g^{(e)} \multimap (h^{(e)} \multimap f^{(t)}) \\ [\textbf{everyone}] & \lambda S.\mathsf{every}(\mathsf{person}, S) : \forall H.(h^{(e)} \multimap H^{(t)}) \multimap H^{(t)}. \end{array}$$

This can be compared, perhaps, to the opposition found in MRS between **ordinary variable** and **handle** arguments of an Elementary Predicate, where the first is dedicated to the intrinsic content of the predicate (as Glue's type e VAR node; cf. substantives and determiners, §3.6), while the second gives information on the scopal behavior (Glue: type t RESTR).

Other restrictions, like MRS's elimination of EP assemblies that lead to empty predicate arguments, are not relevant for Glue, since implicitly present: Backtracking is intrinsic to the notion of proof,[9] so that no invalid meaning can be derived from a valid proof.

[9] It may be seen, in fact, as taking the form of **hypotheses** in a proof; It is therefore connected to the **cut elimination** issue, a major one in Proof Theory.

Multiple proofs. Different proofs, if any, will yield different readings (and thus help handling ambiguity). Let us return to the ambiguous example *Every woman loves a man*, to illustrate scope management:

$$\begin{array}{ll}[\textbf{every-woman}] & \lambda S.\mathsf{every}(\mathsf{woman}, S) : \forall G\ (g \multimap G) \multimap G \\ [\textbf{loves}] & \lambda y.\lambda x.\mathsf{love}(x, y) : h \multimap (g \multimap f) \\ [\textbf{a-man}] & \lambda S.\mathsf{a}(\mathsf{man}, S) : \forall H\ (h \multimap H) \multimap H\end{array}$$

Even upon considering simplified premiss statements (QNP as one statement) [**every-woman**], [**a-man**] and [**loves**], two distinct proofs may be traced:

- After instantiation of variable H to $g \multimap f$, [**every-woman**] and [**loves**] combine (modus ponens and simplification by β-reduction) to yield a subproof of [**every-woman-loves**] := $\mathsf{every}(\mathsf{woman}, \lambda x.\mathsf{love}(x, y))$. The latter combines with [**a-man**] = $\lambda S.\mathsf{a}(\mathsf{man}, S) : (g \multimap f) \multimap f$ to yield the final $\exists\forall$ meaning, *certified* by the final right-hand side (it should be the σ-mapping of the global f-structure):

$$\begin{aligned}[\textbf{every-woman}], [\textbf{loves}], [\textbf{a-man}] &\vdash [\textbf{every-woman-loves}], [\textbf{a-man}] \\ &\vdash \mathsf{a}(\mathsf{man}, \lambda x.\mathsf{every}(\mathsf{woman}, \lambda y.\mathsf{love}(x, y))) : f\end{aligned}$$

- However, as hinted at in the introduction, another derivation would successfully yield f as well, but with a different meaning part. If the *a man* resource h is first discharged *as an hypothesis*, then one may choose to combine [**loves**] and [**a-man**] first, thus giving *every woman* the wide scope ($\forall\exists$):

$$\begin{aligned}[\textbf{every-woman}], [\textbf{loves}], [\textbf{a-man}] &\vdash [\textbf{every-woman}], [\textbf{loves-a-man}] \\ &\vdash \mathsf{every}(\mathsf{woman}, \lambda y.\mathsf{a}(\mathsf{man}, \lambda x.\mathsf{love}(x, y))) : f\end{aligned}$$

The differences appear in the derivation trees, respectively:

$$\dfrac{\dfrac{\lambda Q.\mathsf{a}(\mathsf{man}, Q) : \forall G\ (g \multimap G) \multimap G}{\lambda Q.\mathsf{a}(\mathsf{man}, Q) : (g \multimap f) \multimap f} \quad \dfrac{\dfrac{\lambda P.\mathsf{every}(\mathsf{woman}, P) : \forall H\ (h \multimap H) \multimap H}{\lambda P.\mathsf{every}(\mathsf{woman}, P) : (h \multimap (g \multimap f)) \multimap (g \multimap f)} \quad \lambda y.\lambda x.\mathsf{love}(x, y) : h \multimap (g \multimap f)}{\lambda x.\mathsf{every}(\mathsf{woman}, \lambda y.\mathsf{love}(x, y)) : g \multimap f}}{\mathsf{a}(\mathsf{man}, \lambda x.\mathsf{every}(\mathsf{woman}, \lambda y.\mathsf{love}(x, y))) : f} \tag{3.6}$$

$$\dfrac{\dfrac{\lambda P.\mathsf{every}(\mathsf{woman}, P) : \forall H\ (h \multimap H) \multimap H}{\lambda P.\mathsf{every}(\mathsf{woman}, P) : (h \multimap f) \multimap f} \quad \dfrac{\dfrac{\dfrac{\lambda Q.\mathsf{a}(\mathsf{man}, Q) : \forall G\ (g \multimap G) \multimap G}{\lambda Q.\mathsf{a}(\mathsf{man}, Q) : (g \multimap f) \multimap f} \quad \dfrac{\lambda y.\lambda x.\mathsf{love}(x, y) : h \multimap (g \multimap f)}{\lambda x.\mathsf{love}(x, y) : g \multimap f} \quad y : [h]}{\mathsf{a}(\mathsf{man}, \lambda x.\mathsf{love}(x, y)) : f}}{\lambda y.\mathsf{a}(\mathsf{man}, \lambda x.\mathsf{love}(x, y)) : h \multimap f}\ (\multimap_i)_h}{\mathsf{every}(\mathsf{woman}, \lambda y.\mathsf{a}(\mathsf{man}, \lambda x.\mathsf{love}(x, y))) : f} \tag{3.7}$$

or, shorter, as a combination of statements:

$$\dfrac{\dfrac{[\textbf{every-woman}] \quad [\textbf{loves}]}{[\textbf{every-woman-loves}]} \quad [\textbf{a-man}]}{[\textbf{every-woman-loves-a-man}]} \qquad \dfrac{\dfrac{[\textbf{loves}] \quad [\textbf{a-man}]}{[\textbf{loves-a-man}]} \quad [\textbf{every-woman}]}{[\textbf{every-woman-loves-a-man}]}\ .$$

This last vision of the semantic construction process perhaps gives the eponymous word "glue" its full scope of meaning.

Remark. As implication elimination is the only binary rule in our (limited) proof system (cf. table 3.2), branching precisely amounts to modus ponens. Implication

Implication elimination and **introduction**

$$\frac{t : \varphi \multimap \psi \quad u : \varphi}{t(u) : \psi}\ (\multimap_e) \qquad \frac{\begin{array}{c} x : [\varphi]^i \\ \vdots \\ t : \psi \end{array}}{\lambda x.t(x) : \varphi \multimap \psi}\ (\multimap_i)_i$$

Universal **quantification elimination** and **introduction**

$$\frac{t : \forall X \varphi}{t : \varphi[\psi/X]}\ (\forall_e) \qquad \frac{t : \varphi}{t : \forall X\, \varphi}\ (\forall_i)\ (X \text{ fresh variable})$$

Axiom rule

$$\frac{}{t : \varphi}\ (\text{ax})_i \qquad \text{if } \mathcal{A}_i = \varphi \text{ and } \Lambda(\varphi) = t$$

Table 3.2: The Core Glue fragment—CHI-compliant.

is therefore responsible for making trees wider, and wider trees are combinatorially more likely to offer different readings. However, what actually triggers ambiguity are the scope variables.

To sum up, ambiguity likelihood grows with the number of implication arguments (viz. hypotheses), but is really released only if scope variables are present to match these arguments.

Remark. The equivalence between the two uncurryied LL forms $h \multimap (g \multimap f)$ and $g \multimap (h \multimap f)$ of the verbal predicate involves hypothesising in Implicative Linear Logic (used above too):[10]

$$\frac{\dfrac{\dfrac{h \multimap (g \multimap f) \quad [h]}{g \multimap f}\ (\multimap_e) \quad [g]}{f}\ (\multimap_e)}{\dfrac{h \multimap f}{g \multimap (h \multimap f)}\ (\multimap_i)_g}\ (\multimap_i)_h \qquad \frac{\dfrac{\dfrac{g \multimap (h \multimap f) \quad [g]}{h \multimap f}\ (\multimap_e) \quad [h]}{f}\ (\multimap_e)}{\dfrac{g \multimap f}{h \multimap (g \multimap f)}\ (\multimap_i)_h}\ (\multimap_i)_g \tag{3.8}$$

3.5 The Glue version used here

3.5.1 Core Glue

The **Core Glue fragment** (as defined in [CvG00, pp. 100-101]) consists of formulas from **Implicative Linear Logic** expanded with the universal quantifier. The way connectives behave can be described in natural deduction by the derivation rules present in table 3.2.

[10] In Multiplicative Linear Logic, it would be a trivial identity (cf. (3.9) below), the curryied form of both $(h \otimes g) \multimap f$ being strictly symmetric w.r.t. g and h.

The Axiom Rule will introduce the meaning entries $t : \varphi \quad (t =: \Lambda(\varphi))$ from the syntax-semantics interface, thus introduced as axioms in the proof-theoretical sense (viz. leaves in derivation trees). LL axioms will be indexed: $\mathcal{A} := (\mathcal{A}_i)_i$.

Remark. Rules governing quantification are so limited (and predictable) in their influence that they will not be considered at all in chapter 5: Quantification elimination will be just dropped (assumption 3, §5.2.1), and a pretreatment of Glue axioms will get rid of all quantifiers, so that the fragment of Glue considered as input to the translation will be nearly quantifier-free and thereby purely implicative (to be detailed farther below, §5.2.2).

3.5.2 Structural definitions

Specific parts of the structure of implicative Glue formulas will be distinguished in the translation (§5.3), that have to be defined here.

Definition 6 (Suffixes) $\mathsf{Suf}(\alpha \multimap \beta) := \mathsf{Suf}(\beta) \cup \{\alpha \multimap \beta\}$ *and* $\mathsf{Suf}(a) := \{a\}$ *if* a *is an atom or a variable.* φ *is a* **proper suffix** *of* ψ *if* $\varphi \in \mathsf{Suf}(\psi)$ *and* $\psi \neq \varphi$.

Remark. This definition happens to gather two intuitions: Suffixes of a formula ψ may be defined inductively by a right-hand depth traversal of the tree structure of ψ (always purely implicative after the modifications from section 5.2), but also as the rightmost substrings of ψ that are again formulas (when skipping unneeded parentheses), viz. its suffixes (in the word-theoretical sense).

The next definition on the contrary, admits only the first kind of definition: It also proceeds of a right-hand depth traversal, but this time collecting the other part of the formula (its premiss, whence the name).

Definition 7 (Right-branching *premisses*) $\mathsf{Prm}_r(\alpha \multimap \beta) := \mathsf{Prm}_r(\beta) \cup \{\alpha\}$ *and* $\mathsf{Prm}_r(a) := \emptyset$ *if* a *is an atom or a variable.*

Definition 8 ((Transitive) conclusion) $\mathsf{cc}^+(\alpha \multimap \beta) := \mathsf{cc}^+(\beta)$ *and* $\mathsf{cc}^+(a) := a$ *if* a *is an atom or a variable.*

Remark. The transitive conclusion of a formula is also one of its suffixes, the smallest.

3.5.3 Polarity

Polarisation of subformulas is not a concept necessary to a first understanding of Linear Logic, but it does help us prove important well-formedness properties needed by the translation (chap. 5). It is also used in some advanced literature on Glue Semantics (cf. [GL98, CFvG99]).

Linear Logic and polarity.

The origins. Polarised linear logic originates in the following transformation—which operates in a broader fragment of Linear Logic (necessitating three further linear operators), and the subsequent observations:

$$\psi \multimap \chi := \psi^{\top} \parr \chi \tag{3.9}$$

Even unknowing of the semantics of these operators, this reminds of the classical identity $A \to B \equiv \neg A \vee V$ in the propositional calculus; and indeed corresponding **de Morgan** and **double negation** laws are valid:

$$(\psi \sharp \chi)^\top = \psi^\top \otimes \chi^\top \qquad (\psi \otimes \chi)^\top = \psi^\top \sharp \chi^\top \qquad \varphi^{\top\top} = \varphi$$

A **normal form**[11] is therefore to be obtained by "pushing the negations inward" to the literals via iteration. In the end, the "negated" literals (appearing under $\cdot^\top$) are termed **of negative polarity**, the other ones **of positive polarity**.

The polarity property of a literal is thus of course always *relative to its containing formula* (just as the "polarity" of disjunctive and conjunctive normal forms).

It is then possible, via identity 3.9, to transfer polarity results to the smaller sublogic that *Implicative* Linear Logic is, i.e. to assign polarity to any subformula of an implication.

The polarity of implication is perhaps more easily understood if one decides to "tag" the conclusion and the premiss of *any* implicative formula. With the transformation from above, one notices that every conclusion of *any* subformula is tagged positively, while every premiss is tagged negatively.

It thus becomes a very handy criterion for classifying formulas according to their resources, which is of course the primary concern in Proof Theory: A proof in Implicative Linear Logic proceeds merely by **"matching" between positive and negative resources** (as pointed out under different forms in the literature, cf. [Per00], say). Theorem 1 below is an instance of this idea, found in [Gal91].

The application. The definition of polarity chosen in this thesis is a convenience definition, i.e. our matter here is not what a positive occurrence of a formula really means, just that we be able to map each subformula of an atom to a binary token.

In fact, one could have as well set polarities the other way around! The important point is just to see that they are opposed; We will thus be able to distinguish the formulas that are **premisses** and those that are **conclusions** in a derivation. This distinction relates to the definitions $\mathsf{cc}^+(\varphi)$ and $\mathsf{Prm}_r(\varphi)$ from subsection 5.3.1, and will also be reflected in terms of the structure of the fragments obtained from axiom formulas.

We define polarity top-down and recursively:

Definition 9 (Polarity of the occurrence of a subformula) *Every (occurrence of an)* **axiom** *in a Glue axiom set $\mathcal{A}$ is of* negative *polarity:* $\mathsf{pol}(\mathcal{A}_j) := -$.

If φ is an **implication** $\psi \multimap \chi$*, then the* **conclusion** *has the same polarity as φ, while the* **premiss** *has opposite polarity:* $\mathsf{pol}(\chi) := \mathsf{pol}(\varphi) =: -\mathsf{pol}(\psi)$.

Convention. Abusively, although only when the context is clear, we will speak of polarity for the *formula*, rather than for its particular occurrence.

Two schemes.

Two polarity schemes will be presented and used below:

[11] To be compared with the **disjunctive** and **conjunctive normal forms** of the propositional calculus, for instance.

- One result applicable to any Implicative Linear Logic (theorem 1). It is not committed to any linguistic assumptions whatsoever;
- One strengthening assumption over the Core Glue fragment relying on this result: our **polarity pattern** (assumption 1); It relies on empirical observations made throughout the history of Glue, although it does not seem to apply to multiplicative extensions of Glue, nor to certain linguistic phenomena yet well rendered by Glue (cf. footnote 12).

The theorem. It basically straightens out the intuition that every ILL derivation proceeds in **matching** negative (conclusions) with positive resources (premisses). It appears in different forms in the LL literature ([Per00, Gal91], e.g.):

The following is a result imported from Proof Theory (taken over from [Gal91]) that establishes certain restrictions on polarities of a fragment. It assumes a classically defined notion of polarity (e.g. as in 3.5.3), which is tantamount to that of definition 9 above (possibly modulo an inversion of polarities).

Theorem 1 (Polarity matching) *Every derivation of implicative Linear Logic matches each positive occurrence of each literal with exactly one negative occurrence of it, up to one literal—the conclusion, appearing once more positively than negatively.*

Remark. The proof can be performed for example via proofs nets (see [Gal91]). In fact, the visual intuition of this result is most directly grasped by looking at proof nets and how arrows may or may not connect literals to literals of opposite polarity and inferring the subsequent conditions for literals.

The result is valid for all kinds of proofs, in particular those with non-atomic and multiple conclusions, which we actually are to avoid.

The polarity pattern. This assumption goes a step further by incorporating the linguistic intuition that a meaning entry will either be a **modifier**, a **consumer** or a **producer** of meaning items (cf. [GL98, CFvG99]). This very naturally amounts for Glue to a classification of axioms w.r.t. how they manage resources (cf. §5.4.2 and its relation to the classification obtained in §5.3.4). The three configurations are then mutually exclusive, which will prove technically very helpful in disjunctive cases.

The assumption, which [GL98] say to hold *"in all the glue analyses [they] know of, with one exception"*[12], thus states a stronger version of the theorem, idiosyncratic to Glue derivations: Not only must the LL atoms *match pairwise* according to their polarity, but they must also verify some *unicity pattern*, namely that all must appear:

1. either exactly twice throughout the whole axiom sequence: one positive occurrence in one axiom, and one negative occurrence in another axiom,

[12]The treatment of anaphora by means of contexts constitutes the exception, and is modelled using the **linear conjunction** connector ($\otimes$) from linear logic, and thus falls out of the scope of this paper.

Another exception, perhaps more troublesome, is the case of generic modifier functors introduced in the theory of modification from [Dal01, chap. 10]

2. or exactly twice in any axiom: once positively and once negatively.

It becomes below the core assumption of our formal scaffolding and the warranty that everything functions fine. In particular, it ensures *unicity*[13] of consumers and producers for a resource, whose mere existence or parity would have followed from theorem 1; It is utilised as such in section 5.4.2.

Assumption 1 (Polarity pattern) *For every satisfiable, quantifier-free Core Glue axiom sequence (i.e., quantifiers having been replaced by variables) and for each of its atoms and variables*—up to one[14]*, there are:*

1. *Either one positive occurrence in one axiom and one negative in another axiom,*

2. *or both a positive and a negative occurrence in zero or more axioms.*[15]

This constitutes a huge step from the plain mathematical, though "visually intuitive", theorem, and there we have to rely on (and be thankful for) genuine *linguistic* know-how that the design of Glue axioms implicitly contains.

3.5.4 Axiom normalisation: Exhaustion of premisses

The introduction of **hypotheses** in derivations may appear arbitrary; it seems indeed to make a difference between the two readings of *Every woman loves a man* (3.7) and (3.6), and is used for the uncurrying identity (3.8) as well.

Yet the difference is inessential, and we will make it clear right from the start in systematically hypothesising premisses of LL axioms of a derivation: The premisses of any formulas are then *exhaustively* examined. It does not make any difference at all for the remainder of the proof, since any resource r hypothesised may be immediately reintroduced via $(\multimap_i)_r$ (and β-reduction). But it does help access the different premisses of an axiom *in any order*, as needed:

$$\dfrac{\dfrac{\dfrac{\dfrac{r_1 \multimap r_2 \multimap \cdots \multimap r \quad [r_1]}{r_2 \multimap \cdots \multimap r} \quad [r_2]}{\vdots}}{\dfrac{r}{r_{\sigma(1)\multimap r}}\,(\multimap_i)_{\sigma(1)}}}{r_{\sigma(2)} \multimap r_{\sigma(1)} \multimap r}\,(\multimap_i)_{\sigma(2)}}{\vdots}$$
$$\dfrac{}{r_{\sigma(n)} \multimap r_{\sigma(n-1)} \multimap \cdots \multimap r}\,(\multimap_i)_{\sigma(n)} \quad \text{for any substitution } \sigma \in S_n$$

Section 5.3 will show that in fact, not all quite all of the premisses become hypotheses, but they are recursively analysed anyway. This normalisation step then comes immediately on top of the syntax-semantics interface.

It will help make explicit the merely implicit underspecification of the Glue axiom sequence: Exhausting premisses of Glue formulas will allow to determine

[13] Only modulo unification, though, because of the presence of variables.

[14] The atomic conclusion of the whole set of axioms, corresponding to the global f-structure. This atom shall appear with one more positive occurrence than it has negative occurrences.

[15] The axioms of the first type are coined **skeleton** axioms in [GL98], the other ones **modifier** axioms. This naming hints at the distinction between scopal behavior and pure meaning contribution, also evoked throughout chapter 5.

each axiom's behavior with the others. Since dominance between axioms is expressed as to whether one constitutes a premiss to another (as suggested in the introduction, page 22), the normalised entries will almost directly make premiss resources available—as hypotheses.

3.6 Glue treatment of classic phenomena

We extend here the syntax-semantics interface evoked in section 3.2.2 to account for slightly more complex phenomena, in particular those triggering other kinds of ambiguity (embedded sentences, adjectival).

We will, however, restrict ourselves to the phenomena for which the translation is known to function flawlessly. Glue approaches to more complex phenomena such as, say, anaphora (cf. [Dal99, chap. 11 & 12], [CvG99]) or general-purpose conjunction (cf. [Dal99, chap. 13], [AC02a]), are rather experimental and not clearly convergent, at least following the two state-of-the-art references ([Dal01, Dal99]).[16] Other phenomena, like **raising** and **control**, are known to work, but without rigorous evidence, since they do not fulfill the preliminary assumptions. Yet other ones, like **relative pronouns**, are conjectured to work, though without further investigation.

A thorougher account of determiners and noun phrases. Considering a Quantified Noun Phrase (QNP) as a single entry, as in (3.1), was a simplification. If adjectival ambiguity is to be rendered, too (cf. the ambiguous $\bar{\text{N}}$ *alleged criminal from London*), or ambiguity embedded in prepositional phrases (cf. *Every representative of a company saw (most samples of) a product*), then determiners and nouns should be assigned distinct entries. For instance, as in *every representative*:

$$r \begin{bmatrix} \text{SPEC} & \begin{bmatrix} \text{PRED} & \text{`every'} \end{bmatrix} \\ \text{PRED} & \text{`representative'} \end{bmatrix} \longmapsto r_\sigma \begin{bmatrix} \text{VAR} & \text{vr } [] \\ \text{RESTR} & \text{rr } [] \end{bmatrix}$$

[**every**]	$\lambda P.\lambda Q.\forall x\ (\mathsf{P}(x) \multimap \mathsf{Q}(x)) : (vr \multimap rr) \multimap \forall R\ (r \multimap R) \multimap R$
[**representative**]	$\lambda x.\mathsf{representative}(x) : vr \multimap rr$
[**a**]	$\lambda P.\lambda Q.\exists x\ (\mathsf{P}(x) \wedge \mathsf{Q}(x)) : (vr \multimap rr) \multimap \forall R\ (r \multimap R) \multimap R$

Remark. Another advantage of separating meaning from LL part in Glue axioms is seen here: Different determiners with distinct semantics (meaning part) receive in fact the same LL formula: their *computational contents* are the same. (Likewise, dominance graphs for these two determiners have exactly the same form, only the labels differ.)

Prepositions are just introduced as depending on a noun phrase; the prepositional phrase thus created may modify an $\bar{\text{N}}$ (which consists of a restriction rr

[16] In fact, there is probably no satisfying treatment of anaphora within the mere Core Glue fragment; All approaches to this phenomenon had to consider so-called **contexts**, whose updating must be synchronous with the rest of the proofs. That is, every statement consists of a formula as seen so far, *conjoined* with a context updating and thus utilising linear conjunction, which could be avoided so far.

inputting a variable vr):

$$\begin{array}{ll}
[\textbf{every}] & \lambda P.\lambda Q.\forall x(\mathsf{P}(x) \rightarrow \mathsf{Q}(x)) : (vr \multimap rr) \multimap \forall R\ (r \multimap R) \multimap R \\
[\textbf{representative-of}] & \lambda z.\lambda x.\mathsf{rep\text{-}of}(x,z) : c \multimap (vr \multimap rr) \\
[\textbf{a-company}] & \lambda Q.\exists z(\mathsf{company}(z) \wedge Q(z)) : \forall C\ (c \multimap C) \multimap C
\end{array}$$

where the presence of a modifying $\textsc{obl}_{of}$ node (cf. (3.10)) has triggered the substantive entry's expecting another resource c first. The separation of these three entries then really lets both scopal ambiguities C and R be (under)specified separately, as needed for a successful analysis of *Every representative of a company.*

Proper nouns will be given the same LL formula as QNPs, that is, their semantics shall be raised (unlike in the simplified examples from above) from type e to $\langle\langle e,t\rangle,t\rangle$:

$$[\textbf{Bill}'] := \mathsf{bill} : b \qquad \rightsquigarrow \qquad [\textbf{Bill}] := \lambda P.P(\mathsf{bill}) : \forall B\ (b \multimap B) \multimap B$$

This goes against the Glue literature encountered, but is motivated by sticking to Dominance Constraints' own raising of proper nouns—for the sake of uniformity in our translation (and accordance with semantic theories!). The possible "ambiguity" thereby introduced (because of the extraneous dominance) will just possibly add a few β-equivalent readings to the translation output—no harm there.

After adding similar entries for *company*, *product*, and the **transitive verb** *saw*:

$$\begin{array}{ll}
[\textbf{company}] & \lambda z.\mathsf{company}(z) : vc \multimap rc \\
[\textbf{saw}] & \lambda y.\lambda x.\mathsf{see}(x,z) : p \multimap r \multimap f \\
[\textbf{a}] & \lambda P.\lambda Q.\exists y\ (\mathsf{P}(y) \wedge \mathsf{Q}(y)) : (vp \multimap rp) \multimap \forall P\ (p \multimap P) \multimap P \\
[\textbf{product}] & \lambda y.\mathsf{product}(y) : vp \multimap rp
\end{array}$$

pertaining to the following f-structure:

$$f\begin{bmatrix}
\text{PRED} & \text{'see}\langle(\uparrow \text{SUBJ}),(\uparrow \text{OBJ})\rangle\text{'} \\
\text{SUBJ} & r\begin{bmatrix}
\text{PRED} & \text{'representative'} \\
\text{SPEC} & \text{'every'} \\
\text{OBL}_{of} & c\begin{bmatrix}\text{PRED} & \text{'company'} \\ \text{SPEC} & \text{'a'}\end{bmatrix}
\end{bmatrix} \\
\text{OBJ} & p\begin{bmatrix}\text{PRED} & \text{'product'} \\ \text{SPEC} & \text{'a'}\end{bmatrix}
\end{bmatrix} \tag{3.10}$$

one may derive here *five* readings of this sentence: because the *a company* quantification is embedded in the modifying PP, the enumeration of readings does not follow the mere exponentiality of quantifiers (which would yield $2^3 = 8$ possibilities): some quantifier positions are not valid.

Sentence-embedding verbs. Scopal ambiguity is not exclusive to determiners; consider the "EMBASSY" sentence:

(3.11) Every man believes that a (sleepy) student yawns.

Due to the embedding of the a quantifier in the subordinate proposition, the sentence receives three readings, one of which *de dicto*, the other two *de re*, respectively:

(3.12) (a) $\forall y(\mathsf{man}(y) \rightarrow \mathsf{believe}(y, \exists x(\mathsf{student}(x) \wedge \mathsf{yawn}(x))))$

(b) $\exists x(\mathsf{student}(x) \wedge \forall y(\mathsf{man}(y) \rightarrow \mathsf{believe}(y, \mathsf{yawn}(x))))$

(c) $\forall y(\mathsf{man}(y) \rightarrow \exists x(\mathsf{student}(x) \wedge \mathsf{believe}(y, \mathsf{yawn}(x))))$

They are rendered by the proof trees displayed in figure 3.2, pertaining to the axioms introduced in the right bottom corner.

Modification. Modifiers are as intuitively modelled in Glue as in type-theoretical semantics:

$$\begin{array}{ll} [\textbf{sleepy}] & : (v \multimap r) \multimap v \multimap r \\ [\textbf{student}] & : v \multimap r \\ [\textbf{skillfully}] & : \forall S^{(t)}\ S \multimap S \end{array}$$

Adverbs can thus naturally modify any non-individual (no type e) resource, often the whole f-structure itself. More complex constraints on adverb attachment may, again,be specified using sortal restrictions. We point to [Dal99, chap. 10] for further information on modification. (In particular an substantial treatment of modification is offered there, that differs from most of the accounts found in other Glue literature, for the sake of a greater uniformity, and challenges assumption 1.)

(a)

$$\dfrac{\dfrac{\forall M(m \multimap M) \multimap M}{(m \multimap f) \multimap f} \quad \dfrac{y \multimap m \multimap f \quad \dfrac{\dfrac{\forall S(s \multimap S) \multimap S}{(s \multimap y) \multimap y} \quad s \multimap y}{y}}{m \multimap f}}{\mathsf{every}(\mathsf{man}, \lambda y.\mathsf{believe}(y, \mathsf{a}(\mathsf{student}, \lambda x.\mathsf{yawn}(x)))) \,:\, f}$$

(b)

$$\dfrac{\dfrac{\forall S(s \multimap S) \multimap S}{(s \multimap f) \multimap f} \quad \dfrac{\dfrac{\dfrac{\forall M(m \multimap M) \multimap M}{(m \multimap f) \multimap f} \quad \dfrac{y \multimap m \multimap f \quad [y]}{m \multimap f}}{\dfrac{f}{y \multimap f}\,(\multimap_i)_y} \quad \dfrac{s \multimap y \quad [y]}{y}}{\dfrac{f}{s \multimap f}\,(\multimap_i)_s}}{\mathsf{a}(\mathsf{student}, \lambda x.\mathsf{every}(\mathsf{man}, \lambda y.\mathsf{believe}(y, \mathsf{yawn}(x)))) \,:\, f}$$

(c)

$$\dfrac{\dfrac{\forall M(m \multimap M) \multimap M}{(m \multimap f) \multimap f} \quad \dfrac{\forall S(s \multimap S) \multimap S}{(s \multimap (m \multimap f)) \multimap (m \multimap f)} \quad \dfrac{\dfrac{y \multimap m \multimap f \quad \dfrac{s \multimap y \quad [s]}{y}}{m \multimap f}}{s \multimap m \multimap f}\,(\multimap_i)_s}{m \multimap f}$$

$$\mathsf{every}(\mathsf{man}, \lambda y.\mathsf{a}(\mathsf{student}, \lambda x.\mathsf{believe}(y, \mathsf{yawn}(x)))) \,:\, f$$

[**every**]	$\lambda P.\lambda Q.\forall y(P(y) \to Q(y)) : (v_m \multimap r_m) \multimap \forall M\ (m \multimap M) \multimap M$
[**man**]	$\lambda y.\mathsf{man}(y) : v_m \multimap r_m$
[**believes**]	$\lambda Y.\lambda y.\mathsf{believe}(y, Y) : y \multimap m \multimap f$
[**a**]	$\lambda P.\lambda Q.\exists x(P(x) \wedge Q(x)) : (v_s \multimap r_s) \multimap \forall S\ (s \multimap S) \multimap S$
[**student**]	$\lambda x.\mathsf{student}(x) : v_s \multimap r_s$
[**yawns**]	$\lambda x.\mathsf{yawn}(x) : s \multimap y$
	precombinations:
[**every-man**]	$\lambda Q.\forall y(\mathsf{man}(y) \to Q(y)) : \forall M\ (m \multimap M) \multimap M$
[**a-student**]	$\lambda Q.\exists x(\mathsf{student}(x) \wedge Q(x)) : \forall S\ (s \multimap S) \multimap S$

Figure 3.2: Glue treatment of the EMBASSY sentence.

Chapter 4

Translations between underspecification formalisms

4.1 Introduction

In this chapter we situate the present contribution in the frame of similar works. This implies a formalisation of the concept of **translation between underspecification formalisms**, for it is twofold; It should be:

- A translation between two *semantic* formalisms, i.e., it should provide direct translation for **meaning**.

  Since Montague's works a canonical choice is a higher-order λ-calculus, and thus also for the two formalisms focused on in this thesis; but many different representations are available, including for Dominance Constraints (as hinted at in §2.1) and Glue Semantics (cf. §3.2.3).

- A translation between two formalisms with a notion of **solution**.

  A graphical intuition of how parts of meaning representation correspond between formalisms, like that sketched in figure 1.2 may constitute a good starting point: But a formal investigation should go beyond the optical similarity. In particular, it should be ensured that readings output by the translation are indeed valid readings (**soundness**), and maybe also that all expected readings are present (**completeness**). This may be achieved only under formalisation of how readings are derived from the concise underspecified representation, hence the notion of solution (e.g. of a dominance constraint, a proof in Glue Semantics, a configuration in MRS, etc.).

Further restrictions on the solutions obtained may be demanded, depending on the translation, as a theoretical guarantee the translation works (theoretical assumptions), or as a condition the translation in usable *in practice* (empirical assumptions).

We will see that in the case of Dominance Constraints as output, proofs will be restricted to a particular fragment of Dominance Constraints, namely the **hypernormally connected** constraints (so-called **nets**, to be defined below). Nevertheless, such a restriction is conjectured to be always fulfilled in constraints generated by linguistic syntax-semantics interfaces: this is called the **Net Hypothesis** ([FKNT04]). One notices thus the relevance of translations between

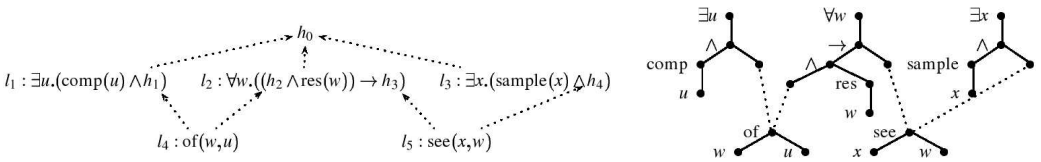


Figure 4.1: Hole Semantics and Dominance Constraints representations for *Every researcher of a company saw a sample.*

underspecification formalisms, especially to Dominance Constraints: Each translation whose output is proved to consist of nets contributes a bit more to an empirical confirmation of the Net Hypothesis.[1]

We will first look at such similar translations already present in the literature: making the connection between underspecification in Dominance Constraints and, respectively, Hole Semantics (§4.2) and Minimal Recursion Semantics (§4.3). We will also take a quicker look at yet other similar works (§4.4).

4.2 From Hole Semantics to Dominance Constraints

Hole Semantics ([Bos96, Bos02])[2] is a formalism defined, as Dominance Constraints, over arbitrary meaning languages, whose formulas are equipped with **holes**, into which other formulas can be **plugged**. Underspecification is realised in that several admissible pluggings are stated via scope constraints. Several actual pluggings may satisfy the constraints; hence the concept of solution mentioned in the introduction is instantiated here as a plugging satisfying the constraints.

A translation from holes to holes and labels to roots can be graphically grasped (cf. figure 4.1). It is even a back-and-forth translation, but only under certain restrictions. These motivate the introduction of the Net Hypothesis.

4.2.1 Hole Semantics

The representations are called **underspecified representations** (USR), which consist of a finite set L_U of labelled formulas $l : F(h_1, \ldots, h_n)$ and a set of constraints C_U on pluggings into holes. **Labels** like l and **holes** like $h_1, \ldots, h_n$ undergo constraints of the form $l \leq h$: "hole h **outscopes** label l".

These constraints are displayed in the graphical representation of USRs as dotted edges, where vertices are then formulas of the meaning language (cf. figure 4.1, left).

A well-formed, so-called **proper**, USR should have:

1. a unique **top element**, that may reach all other nodes in the graph;

2. an acyclic graph;

[1] An empirical study conducted on HPSG corpuses ([FKNT04], developed in[FKT05]) already showed its relevance and pointed at mistakes in annotation or at incoherences in the syntax-semantics interface.

[2] We rely here on the version of Hole Semantics described in [KNT03], which corresponds to the newer original source [Bos02].

3. the property that every label and every hole occurs exactly once in L_U, except for the top hole.

Resolution of an USR amounts to plugging labels into holes, while respecting the constraints. Formally, a **plugging** is a bijection from the holes to the labels. But an **admissible plugging** P for a proper USR must also fulfill scope constraints, namely: $k \leq k' \in C_U \implies k'$ P-dominates k, where P-domination is defined below:

Definition 10 (P-domination) *If k and k' are holes or labels of some USR U, P a plugging on U, then k* **P-dominates** *k' if either:*

1. *$k : F \in L_U$ and k' occurs in F, or*
2. *$P(k) = k'$ if k is a hole, or*
3. *there is a hole or label k'' s.t. k P-dominates k'' and k'' P-dominates k'.*

4.2.2 Translation

A straightforward translation has been exhibited in [KNT03] from proper USRs to normal dominance constraints:

labeled formula $l : F(h_1, \ldots, h_n)$	$\longmapsto$	labeling constraint $l : F(h_1, \ldots, h_n)$
constraint $l \leq h \in C_U$ ($h \neq$ top hole)	$\longmapsto$	dominance constraint $h \triangleleft^* l$
distinct labels l and l'	$\longmapsto$	inequality constraint $l \neq l'$

Labels and holes of the input USR thus become, respectively, roots and holes of the output constraint, as shown in figure 4.1.

One notices that the constraints obtained are in fact *compact* (cf. page 14), since all labeled formulas introduce holes at depth one. In fact, [KNT03] gives a back-and-forth translation from a proper USR to a compact normal dominance constraint C, together with proofs of the preservation of their solutions: Pluggings of U and *constructive* solutions of C correspond.

Here we pause and notice that only constructive solutions come in question. They are indeed what one is looking for in solving linguistically ambiguous constraints: solution trees with no extraneous material.[3]

But the satisfiability/enumeration algorithms for Dominance Constraints enumerate solely the **solved forms** of input constraints (cf. §2.3.3, footnote 12). So the link between solutions and, with it, the gain of efficiency in solving is not clear yet between Hole Semantics and Dominance Constraints.

In the same paper, however, it is proved that under certain well-formedness conditions, the notion of satisfiability for constructive solutions and solved forms coincide.

The final theorem states: Every solved form of a hypernormally connected (defined below), leaf-labeled normal dominance constraint has a constructive solution.[4]

[3] As a counterexample, the left constraint on figure 4.2 admits no constructive solution; the tree on the right is a solution, yet a node not explicitly stated as a label (g) had to be added.

[4] Actually, in place of "hypernormally connected" is stated another property: "chain-connected", but these notions are equivalent for normal constraints ([Kol04, chap. 6]). Leaf-labeledness is a property empirically verified in linguistic syntax-semantics interfaces ([KNT03]).

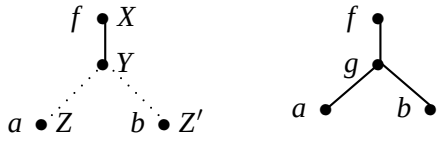


Figure 4.2: A dominance graph that is not hypernormally connected, and a solution thereof. The only edges connecting hole Y are dominance edges. The solution is not constructive—g added.

This closes, in a restricting fashion, the issue pointed at in the introduction of a translation with conservation of solutions: When considering only hypernormally connected constraints, the enumeration/satisfiability algorithms from DC may be used.

4.2.3 Towards the Net Hypothesis

The empirical restriction to the translation just mentioned is defined here.

Definition 11 (Hypernormal connexity) *A* **path** *in a dominance graph is* **hypernormal** *if it does not comprise two incident dominance edges out of a non-root node.*

Constraint C is hypernormally connected—or C is a **net***—if there is an hypernormal path between every two nodes of C.*

Now we see under a different light what was wrong with the constraint from figure 4.2: it is not hypernormally connected, since the only path from Z to Z' uses two adjacent dominant edges, both out of hole Y.

But it is strongly believed, and the next section will make it yet clearer, that the theoretical restriction of considering only hypernormally connected constraints is not a restriction *in practice*: For linguistic purposes, we will in fact *always* encounter nets. This is the statement of the following conjecture.

Conjecture 2 (Net Hypothesis) *All dominance graphs issued from linguistic syntax-semantics interfaces are nets.*

4.3 From Minimal Recursion Semantics to Dominance Constraints

Minimal Recursion Semantics ([CFS01], algebraically and uniformly specified in [CLF01]) is a semantic formalism originally developed for HPSG, although it has proved, even more than Glue, really autonomous, as a "flatter semantics" formalism. The stress there lies on the efficiency of semantic construction, and it could thus be successfully incorporated in large-scale grammars (as the English Resource Grammar [CF00]).

But more importantly for us, it works with low-depth semantics. To this extent, it resembles much compactified dominance constraints, or even perhaps Hole

Semantics; The eponymous "minimal recursion" constrains the MRS representation in a way that no logical predicate have a recursion depth of more than one. Quantifiers are a particular type of predicates, which appear to be "floating" between pieces of meaning (cf. figure 4.3, left), whence ambiguity underspecification is realised. A particular reading, or **configuration**, is then a choice of where quantifiers shall attach.

Some design peculiarity, however, prevent a "naive" translation to be used in practice: MRS' conjoining of predicates must be somehow accommodated, hence a two-step translation.

4.3.1 Minimal Recursion Semantics

MRS is a constraint language on formulas defined over a vocabulary of

1. Metalanguage variables, called **handles**;
2. Meaning language variables, that is, metalanguage constants, split into individual variables $x, y, z, \ldots$ and event variables $e, e', \ldots$;
3. Functions symbols P, among which binary quantifier symbols Q_x are distinguished;
4. The **qeq** relation symbol $=_q$.

Literals of MRS formulas consist of **elementary predications** (first two), with handle and argument positions (resp. left and right of the :), and **handle constraints** (third one):

1. $h : P(x_1, \ldots, x_n, h_1, \ldots, h_m)$: with $n, m \geq 0$
2. $h : Q_x(h_1, h_2)$
3. $h_1 =_q h_2$

Sets of such literals yield well-formed representations (called MRS as well) if they further verify:

1. Every handle occurs at most once in argument position;
2. Every handle constraint $h =_q h'$ relates a handle argument h to a label h';
3. Every individual variable x in argument position induces a unique literal $h : Q_x(h_1, h_2)$ that quantifies over x.[5]

MRSs admit a straightforward graphic representation by encoding EPs $h : P(x_1, \ldots, x_n, h_1, \ldots, h_m)$ into solid directed tree edges $(\mathsf{P}_{x_1,\ldots,x_n}, h_1), \ldots, (\mathsf{P}_{x_1,\ldots,x_n}, h_m)$, and implicit binding constraints in explicit dotted edges (unless transitively redundant). See for example the left picture on

[5] This version (taken over from [FKNT04]) slightly differs from the original definition ([CFS01]), where MRSs are seen as triples $\langle h_0, \{\langle \text{EPs} \rangle\}, \{\langle \text{handle constraints} \rangle\}\rangle$, with h_0 being distinguished as the **top handle**. However, top handles are only relevant to MRSs with unconnected graph (cf. [BDNM04]), which is reasonably enough never the case in linguistic applications.

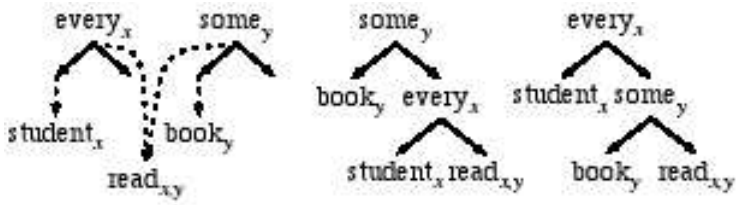


Figure 4.3: Graphical representation of an MRS and its two configurations for *Every student read a book.*

figure 4.3 that represents, without scope commitment, *Every student read some book*, whose MRS would be as following:

$$\{h_5 : some_y(h_6, h_8), h_7 : book(y), h_1 : every_x(h_2, h_4),$$
$$h_3 : student(x), h_9 : read(x, y), h_2 =_q h_3, h_6 =_q h_7\}$$

Without going further into details, one may guess what resolution of ambiguity amounts to:

- Connecting all handles pairwise with arguments, whereat ensuring that
- All quantifier EPs Q_x **outscope** all EPs containing their x (outscoping is defined in an intuitive, similar way as P-domination above);
- EPs do not outscope themselves;
- All EPs are connected;
- Handle constraints are satisfied: $h =_q h' \Longrightarrow h$ outscopes h'.

Such solutions to an MRS M are called **configurations** of M, or **scope-resolved** MRSs (they are MRSs of a certain form, with no pending handle constraints). For instance, figure 4.3 represents an MRS (left) for the ambiguous sentence *Every student read some book* and the two configurations it admits (right). These obviously correspond to the two readings (resp. $\exists\forall$ and $\forall\exists$) of the sentence.

We already encountered **conjunction** in MRSs implicitly: all constraints from the set M are understood as conjoined (Dominance Constraints, e.g., would make the conjunction explicit). But conjunction may also appear, necessarily, in configurations of an MRS M which does not involve any conjunction itself. (Cf. figure 4.4: any solution there necessarily implies P_3 and P_2 be conjoined as $\{P_2, P_3\}$.) Configurations involving (implicit) conjunction of EPs are called **merging configurations**.

W.r.t. the Net Hypothesis, a translation would more easily accommodate such MRSs as in fig. 4.4 as *unsolvable*, but this is not the case.

$\{h : P_1(h'), h' : P_1, h' : P_2\}$

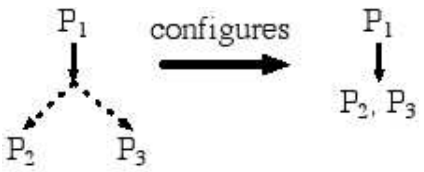


Figure 4.4: An MRS that is no MRS-net (left). A solution necessary involves conjunction (right).

4.3.2 Translation

MRSs as in figure 4.4 represent a problem. As it happens, the shape of the MRS graph looks familiar: it reminds of that of the dominance graph from figure 4.2 above, which is not hypernormally connected.

The analogy is not fortuitous, and such MRSs are called MRS-non-nets: their translation onto dominance constraints (see below) would yield non-nets. Whereas Dominance Constraints accommodates them in introducing extra material and producing non-constructive solutions, MRS necessitates merging configurations.

But again, the Net Hypothesis allows then to forget about such structures. In fact, the second important result in [FKNT04] is that the merging semantics of MRSs can be skipped altogether, *for nets*. Conjunctions are eliminated in a preliminary step, and nets will never yield further conjunctions in their solutions. The translation is therefore a two-step process:

Resolving conjunctions. The following rule is exhaustively applied to MRS M, involving a new binary function symbol & and two fresh variables:

$$h : E,\ h : E' \implies h : \&(h_1, h_2),\ h_1 : E,\ h_2 : E'$$

The depth 1 property is lost through this process, so MRSs shall be compactified afterwards (where compactification is defined similarly as for dominance constraints).

Translating and normalising. After this pre-normalisation step, a translation from MRS literals onto dominance constraints consists of the following:

EP $h : P(x_1, \ldots, x_n, h_1, \ldots, h_m)$	$\longmapsto$	labeling constraint $l : F(h_1, \ldots, h_n)$
quantifier EP $h : Q_x(h_1, h_2)$	$\longmapsto$	labeling constraint $h : Q_x(h_1, h_2)$
handle constraint $h =_q h'$	$\longmapsto$	dominance constraint $h \lhd^* h'$
$h : Q_x(h_1, h_2)$ and $h_0 : P(\ldots, x, \ldots)$	$\rightsquigarrow$	dominance constraint $h \lhd^* h_0$
handles h and h' in distinct label positions	$\rightsquigarrow$	inequality constraint $h \neq h'$

Assessment. The important statement of the paper is that the translation's first step is no restriction, provided only nets are considered. The successive proofs involve, again (cf. §4.2), the crucial property that solved forms and constructive solutions coincide on nets.

As a conclusive interrogation to the "net issue": An empirical study conducted on outputs of ERG outputs—MRSs, then—originally in order to test the translation pointed at mistakes and incoherences in the outputs: an analysis of the non-nets showed that they were either false or genuine non-nets that are incomplete ([FKT05], cf. also [FKNT04]). This led to considering hypernormal connectivity as a linguistically reliable condition: "All linguistically correct expressions are nets." Hence, could the Net Hypothesis be used as a "safety criterion" for syntax-semantics interfaces in the future?

4.4 Other examples

Several other translations between underspecification formalisms, connecting somewhat older formalisms, are found and investigated in the literature.

From QLF to Hole Semantics. Quasi-Logical Forms ([Als92]) are one of the first underspecified semantic formalisms. However, [Lev05] remarks that underspecification, more precisely, the dominance relation, is only implicitly stated in QLF.

The author thus proposes a translation that "decouples" dominance from meaning relations in sentence representations, so as to make dominance explicit and directly expressible as HS pluggings.

The gain in translating between formalisms is explicitly stressed: it is argued that QLF representations are more intuitive and easier to construct, while Hole Semantics makes ambiguity more apparent.

From f-structures to UDRSs and QLFs. Underspecified Discourse Representation Theory ([Rey93]) is yet another a pioneering formalism realising underspecification. Its representations rely on Discourse Representation Theory ([KR93]), and expand its objects by stating scope relations. In [vGC99] two back-and-forth translations are exhibited from LFG's f-structures (cf. §3.2.1) onto, respectively, UDRSs and QLFs.

The translations take advantage of the peculiar layer structure of LFG, in that it uses the higher-level, syntactically underspecified f-layer (as opposed to strongly hierarchical structures (e.g. trees) usually needed to steer composition in syntax-semantics interfaces). The translation is a *direct* mapping of LFG's objects onto UDRT (resp. QLF) semantically underspecified representations (i.e. not relying on an actual semantic layer, like Glue).

Chapter 5

A translation from Glue Semantics to Dominance Constraints

5.1 Introduction

The following chapter presents the theoretical work achieved towards a working translation form Glue Semantics to Dominance Constraints.

In a nutshell. The main ideas of this chapter are that we will:

1. Translate Glue derivations trees into trees over a language of Glue rule names;
2. Translate Glue axioms' logical dependencies on one another into dominance constraints;
3. Identify every solution of the output dominance graph with a Glue derivation.

To illustrate these three points on an example, let us return to the ambiguous sentence referred to in the previous chapters: *Every woman loves a man.* The input

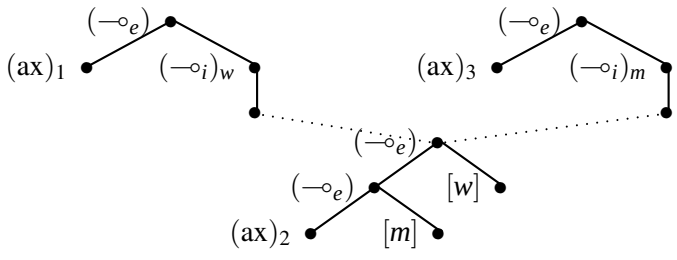


Figure 5.1: Translation output for *Every woman loves a man.*

$\mathcal{A}_1$	[**every-woman**]	$\lambda S.\mathsf{every}(\mathsf{woman}, S) : (w \multimap W) \multimap W$
$\mathcal{A}_2$	[**loves**]	$\lambda y.\lambda x.\mathsf{love}(x, y) : m \multimap (w \multimap f)$
$\mathcal{A}_3$	[**a-man**]	$\lambda S.\mathsf{a}(\mathsf{man}, S) : (m \multimap M) \multimap M$

$$\dfrac{\dfrac{}{(w \multimap W) \multimap W} \quad \dfrac{W}{w \multimap W}}{W}\ \mathcal{A}_1 \qquad \dfrac{\dfrac{\dfrac{}{m \multimap (w \multimap f)}\ (\mathrm{ax})_2 \quad [m]}{w \multimap f} \quad [w]}{f}\ \mathcal{A}_2 \qquad \dfrac{\dfrac{}{(m \multimap M) \multimap M} \quad \dfrac{M}{m \multimap M}}{M}\ \mathcal{A}_3$$

Table 5.1: Translation input for *Every woman loves a man* (simplified version: QNPs are considered as one block).

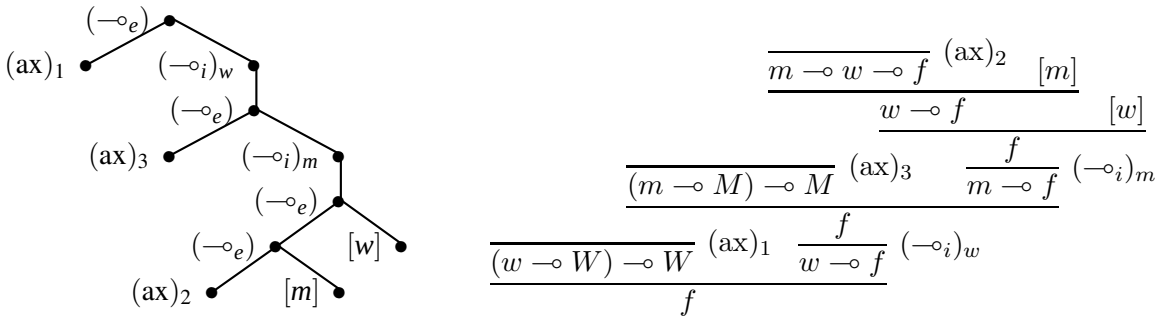


Figure 5.2: Correspondence between output solutions and Glue derivations. The $\forall\exists$ reading of *Every woman loves a man* is considered here.

to consider will consist of the Glue axioms, displayed and exhaustively decomposed (cf. §3.5.4) as in table 5.1.

As for the output, it would be expected to have at least the dominance structure of the graph from figure 2.2, that is, one would like it to admit exactly two constructive solutions, respectively corresponding to the $\forall\exists$ and $\exists\forall$ readings.

In fact, the output will have the form depicted in figure 5.1: **Labels** of the dominance graph are Glue **rule names**, so that the fragments' structures are exactly those of the corresponding derivations; Fragment **leaves** correspond to the "leaves" of a derivation tree, i.e. to the use of either **axiom rules** or **hypotheses**.

Assumed intermediate results in subproofs (here the $\mathcal{A}_1$ (resp. $\mathcal{A}_3$) supposes a proof of W (resp. M) be available) correspond to holes (unlabelled nodes). **Dominance** arises from that **dependence on an intermediate result**. Here variables W and M may both instantiate to f, so that the dominance behavior is exactly that of figure 2.2.

The question remains of how to accommodate the solution concept (cf. §4.1) to this output dominance graph: this is the **soundness** issue (and in the ideal case, the completeness issue as well). Because: what do solutions of the output graph represent? How are they related to Glue's notion of solution? It suffices to remark that any constructive solution of the output graph may be one-to-one mapped onto a derivation tree, as suggested in figure 5.2.

Before the translation. One may notice in the derivation tree that two derivation steps do not correspond to any specified in Core Glue (table 3.2). For instance:

$$\frac{(w \multimap W) \multimap W \quad w \multimap f}{f}$$

is strictly speaking not a modus ponens application; it actually consists of two steps condensed in one rule: Instantiation of variable W to f (i.e., $(\forall_i)_{[f/W]}$-application) and *afterwards* modus ponens ($(\multimap_e)$-application) as f inferred from $(w \multimap f) \multimap f$ and $w \multimap f$. This rule is called **modus ponens modulo unification**: $(\multimap_e)_u$, and is introduced in section 5.2.2, as well as the justifications for getting rid of the quantifiers (Glue variables stand free in our translation's input).

The correspondence between the solutions of the output graph and the derivation trees (resp. left and right in figure 5.2) is made explicit in section 5.2.3, where modified versions of the notion of derivation tree are introduced, tailored for our translation's output. In particular, the correctness of a derivation tree is redefined, so as to take into account the new modus ponens introduced.

A few empirical observations in the Glue literature, about what form the axioms provided by the syntax-semantics interface may take, allow us to make some assumptions on the input axioms, useful for later uses. They are exposed in section 5.2.1.

Rules as labels. Thereafter, section 5.3 introduces the translation, in two steps: outputting firstly plain **fragments** of Dominance Constraints (§5.3.1) and secondly the actual **dominance** constraints between these fragments (§5.3.4).

From the fragments alone, several useful properties can already be extracted (§5.3.2 and §5.3.3), which are meant to converge towards one result, the main contribution—soundness.

Architecture of soundness. As evoked above, the translation is only satisfying insofar as the notion of solution between the two formalisms is translated as well; thus we will have to ensure that every solution of the output graph is indeed a correct derivation tree as redefined in §5.2.3—that the translation is **sound**.

The proof of soundness itself may be decomposed along the steps illustrated in figure 5.1: The properties are all embedded in one another, because every step further in the proof demonstrates one more general property of the output graphs.

For instance, we call *local* soundness a satisfying behavior of the *fragments* output, with no statement yet about dominance. This can be proved already after introducing the mere fragments (§5.3.3).

For the solutions of the output graphs to be correct derivations trees, they have to : firstly, *pass up the right formulas* at each rule; but also, secondly, *keep track of hypotheses* made throughout the derivation: A $(\multimap_i)_\varphi$-rule may only be applied if a so far unused hypothesis φ has been made up the derivation tree.[1]

Weak soundness refers then to output solutions which fulfill only the former, and thus are possibly incorrect w.r.t. to hypothesis book-keeping. Soundness

[1] And in Linear Logic it is also important that there be not *too many* hypotheses made either (unlike First-Order Logic): All hypotheses made must thus be matched *pairwise* by $(\multimap_i)$-rules, hence a careful book-keeping of them is needed.

will be only *partial* when one is only able to ensure that parts of a solution tree are well-formed, but not the tree as a whole (for example if it keeps too many hypotheses).

Important results inherited from the **polarity** aspect of Linear Logic are also cited and used in §5.4.2, and a well-formedness result w.r.t. Dominance Constraints (§5.4.1) is brought: the output dominance graphs are there proved **normal**. The last steps of the soundness proof are made in the final section (§5.4.3).

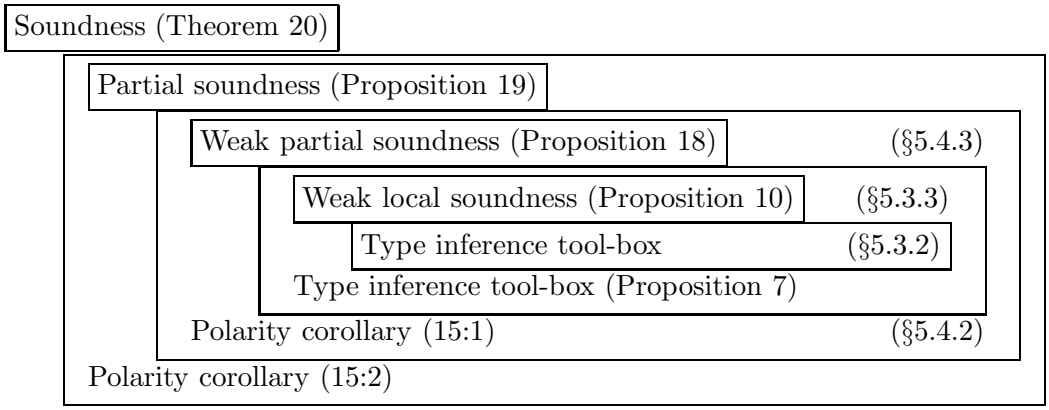


Figure 5.3: Proof architecture for soundness.
Every boxed property is proved using results in the adjacent rectangle; All boxed properties of the translation thus chainwise imply each other: Soundness ⇒ Partial soundness ⇒ Partial weak soundness ⇒ Weak local soundness.

The closing section (§5.5) hints at what misses for a completely satisfactory translation, among which **completeness**, and shows how this is nevertheless *empirically* verified.

5.2 Formal preliminaries

The following section presents all the material necessary to introduce the translation itself: theoretical assumptions issued from empirical observations (§5.2.1), accommodation of quantifiers in Glue formulas, and redefinition of derivation trees.

Convention.

- We will the term "derivation (tree)" as a synonym for "proof (tree)".
- Unless there is need to make it explicit, $\mathcal{A} = \mathcal{A}_1, \ldots, \mathcal{A}_n$ will designate the sequence of all axioms for the given sentence to be analysed. Although actual Glue axioms comprehend both sides of the CHI (cf. §3.2.3), all the work in this chapter relies on the sole LL parts (right-hand sides) of these. Thus, LL formulas will be implicitly understood in $\mathcal{A}$.
- Besides, unless otherwise stated, the LL formulas considered in the following are taken among the **Core Glue Fragment** (defined in table 3.2 above), after undergoing the small modifications described in subsection 5.2.2.

The need for such modifications has motivated the theoretical assumptions from the next section.

5.2.1 Theoretical Assumptions on Glue

As for the linguistic phenomena to be accounted for, we restrict ourselves to those exposed in chapters 6–10 from [Dal99]. This is particularly important insofar as the following assumptions rely, from the empirical side, on the treatment of those phenomena observed in the Glue literature (as presented in §3.6 above).

The following assumptions restrict the types of Core Glue derivations which we will consider as input to the translation. We argue that, from an empirical point of view, they do not constitute a restriction, though, and show most make linguistic sense, too.

Assumption 3 *No $\forall$-introduction step is ever used.*

Remark. We conjecture there exists a simple proof-normalising method to eliminate all $(\forall_i)$-rules from any Glue derivation. It would allow us not to merely *assume* the $(\forall_i)$-free form of derivations, but to *prove* this can be obtained systematically; that is not going to be done in this thesis.

The following assumption is needed to get rid of quantifiers (cf. next section).

Assumption 4 *No $\forall$ quantifier is ever left-embedded. That is, any $\forall$ quantifier appearing in an implicative formula $\varphi = \psi \multimap \chi$ must appear in the right-hand term χ of the implication.*

Since linear implication in Glue axioms means *dependency* on a particular resource, that means that no semantic predicate should itself put a scope restriction on one of its arguments, but only these arguments themselves may.

For instance, the entry for a quantifying determiner, as for the LL part, takes the form

$$\forall S\ (s \multimap S) \multimap S,$$

where S is the scope variable, modified by the noun resource s. Were there a variable T quantified within the parenthesis, as in, say, $(\forall T\,(s \multimap T) \multimap S) \multimap S$, the final resource passed upward in the LFG σ-structure: S, would overlook altogether its argument s; in $(\forall T\,(T \multimap s) \multimap S) \multimap S$, the final scope resource S would only have a dummy argument T; In any way, any potential new scope information induced by T is lost outside its quantifier's scope.

The purpose of quantifiers in Glue is to allow free instantiation of scope variables (S) and then just pass forward this scope information—possibly modified (as $s \multimap S$), in which case it depends on one or several arguments (here s). But it would make no linguistic sense not to reuse the scope information, which would be the case if the arguments themselves bore their own scope (as T in $(\forall T\ \Phi(T)) \multimap S$).

Assumption 5 *There is at most one variable in any axiom.*

Remark. This assumption is purely technical, and it might *a priori* well happen that two variables or more be needed for the treatment of a particular linguistic phenomenon. Nevertheless, we at present do not know of any such phenomena. It is, besides, probably implied by the polarity pattern defined above (assumption 1).

Assumption 6 *Formulas appearing as φ in rules $(\multimap_i)_\varphi$ do not contain any variable.*

Remark. Again, we conjecture this to be provable, actually: to be accommodated by some proof-normalising step.[2]

Yet another assumption is needed in the definition of the output fragments.

Assumption 7 (Conclusion of type t) *For any axiom sequence $\mathcal{A}$, $\mathsf{cc}^+(\mathcal{A})$ only consists of atoms of type t or variables.*

This is arguably a purely linguistic assumption (definition 19 could probably be modified to handle any type of conclusion); it has no influence on the structure of the translation. But it is coherent with the treatment of noun phrases in Montague Semantics ([Mon74]) and the fact that scope variables are assigned type t in the Glue literature (cf. sortal restrictions in §3.4, page 29). [3]

5.2.2 Glue and quantifiers

In Glue derivations as shown in the literature, substitution of variables (only *universally* quantified in the Core Glue Fragment) and *modus ponens* (i.e. the application of, respectively, $\forall$-elimination and $\multimap$-elimination) are mostly performed in one step instead of two, often for the sake of displayability of the derivation trees. This we also want our input formalism to do, which led to the formal introduction of a simplification rule, a modus ponens tailored for direct substitution: a $\multimap$-elimination rule *modulo unification* ($(\multimap_e)_u$).

It relies on there not being any quantifiers in the input formulas. We thus translate every Glue axiom the following way: All quantified variables appearing in axioms are replaced by *pairwise distinct*, *new* symbols (**variable symbols**; there is thus only a finite number for any axiom sequence).

Any of these newly introduced variable symbols $S_1, \ldots, S_k$ may be instantiated by $(\multimap_e)_u$ application to any other Glue formula from $\mathcal{L}$ (including those containing other S_i symbols as well).

[2] An idea would be that any candidate, say, $(\multimap_i)_{\Phi(G)}$, necessary implies an hypothesis $[\Phi(G)]$ up the derivation tree. But the variable G introduced by the rule will be eventually instantiated, say, to φ. It probably amounts to the same (yields equivalent proofs), then, to *directly* hypothesise the instantiated form: $[\Phi(\varphi)]$, since $\Phi(G)$ subsumes (is more general than) $\Phi(\varphi)$:

$$\dfrac{\dfrac{\begin{array}{c}[\Phi(G)]\quad\cdots\\ \vdots\\ \psi\end{array}}{\Phi(G)\multimap\psi}\,(\multimap_i)_{\Phi(G)}}{\Phi(G)\multimap\psi}\,[\varphi/G] \quad\rightsquigarrow\quad \dfrac{\begin{array}{c}[\Phi(\varphi)]\quad\cdots\\ \vdots\\ \psi\end{array}}{\Phi(\varphi)\multimap\psi}\,(\multimap_i)_{\Phi(\varphi)}$$

[3] The notable exception to the assumption are the axioms for **proper nouns** found in the Glue literature (e.g., [Dal01]), which consist of a type e atom. Hereto the solution is to raise them, as we saw in §3.6, which exactly corresponds to the treatment in Montague Semantics. (Another solution could consist in introducing a special case of definition 19 for axioms consisting of a single type e atom.)

We say that formula φ **subsumes** ψ if there is a substitution of variables to formulas that maps φ to ψ.[4] We denote by $\mathsf{mgu}(\psi, \psi')$ the **most general unifier** of ψ and ψ'. It is the most general (i.e. smallest, in terms of number of mapped variables) substitution of variables of both ψ and ψ'.

The subsequent theoretical argument is that any derivation obtained this way is equivalent to the original Glue derivation in two steps:

Quantifier elimination

Every axiom Φ involving a variable (unique under assumption 5) undergoes the following customised $\forall$-elimination step:

$$\frac{\Psi(\forall X\, \Phi(X))}{\Psi(\Phi(S))} \quad \text{for any variable } X \text{ and } S \text{ a fresh variable symbol,}$$

where Ψ thus substitutes all occurrences of its (universal) quantifier with quantifier-free $\Phi(S)$ and replaces the formerly bound variable with a fresh one.

In the Glue literature, these substitutions would be performed just before an application of $(\multimap_e)_u$ (so as to stick to the Core Glue schema $(\forall_e)$-$(\multimap_e)$). However, the order in which metavariables are introduced to the axiom is irrelevant: Thanks to the identity (true in Linear and in First-Order Logic):

$$\varphi \multimap \forall P\, \psi(P) \equiv \forall P\, \varphi \multimap \psi(P) \qquad \text{if } P \text{ does not appear free in } \varphi, \tag{5.1}$$

all embedded quantifiers can be extracted "in-outward" in the structure of all axioms, thus allowing the quantifier-elimination process to be performed *prior to* any actual proof.

Assumption 4 ensures that this elimination does not interfere with modus ponens; assumption 6 ensures it does not interfere with $(\multimap_i)$ either: they imply that identity (5.1) covers all cases of quantifier extraction.

This was but the first step of the process, for now these new variable symbols must be accommodated, more accurately: instantiated, in the modified modus ponens, just the same way as $\forall$-elimination is performed in Core Glue.

Modus ponens with built-in unification

Every **modus ponens** (or $(\multimap_e)$, or $\multimap$-elimination) step is replaced by a modus ponens *modulo unification*:

$$\frac{\psi \multimap \chi \quad \psi'}{\mathsf{mgu}(\psi, \psi')(\chi)}\ (\multimap_e)_u \qquad \begin{array}{r}\text{if } \psi \text{ subsumes } \psi',\\ \text{(in which case } \mathsf{mgu}(\psi, \psi') \text{ is defined),}\end{array}$$

where $\mathsf{mgu}(\psi, \psi')$ leaves ψ' invariant, since ψ' is subsumed by ψ.

[4] A substitution as a formula mapping $f : \mathcal{L} \to \mathcal{L}$ is just the function canonically induced by the actual substitution of a set $\mathcal{T} \subseteq \mathcal{L}$ of variables $\sigma : \mathcal{T} \to \mathcal{L}$: For every formula $\varphi \in \mathcal{L}$, $f(S) := \sigma(S)$ if $S \in \mathcal{T}$ and $f(\varphi \multimap \psi) := f(\varphi) \multimap f(\psi)$. Both will be called substitutions in an unambiguous context.

Convention. Henceforth, for convenience, all the new variable symbols (from now on short: **variables**) and only these will be designated by capital Roman letters in Glue formulas.

Generic Glue formulas will be designated by Greek letters $\varphi, \psi, \chi, \ldots$ and atoms that are no variables by lower-case Roman letters: $a, b, f, g, \ldots$

5.2.3 Modelling derivation trees

In this subsection we adapt the notion of **derivation tree** to solutions of dominance graphs (as shown in figure 5.2, left) and thereby formalise what is intuitively performed upon verifying a proof tree.

This process not only consists in local applications of rules (thanks to quantifier elimination: only $\multimap$-elimination, $\multimap$-introduction, axiom and hypothesis introduction), but also in global *book-keeping* of formulas, in that one has to keep track of all hypotheses introduced at each node (to determine whether $\multimap$-introduction is valid). Since the same hypothesis may a priori be introduced more than once before use in the same derivation tree, the book-keeping information must be more than binary (hypothesis φ either bound or free): it must include the actual *number* of occurrences of φ introduced and not yet bound.

For this specific purpose we introduce the handy notion of **multisets**[5], an intuitive generalization of classic sets.

Definition 12 (Multisets) *A* **multiset over the base set** X *is a function* $X \to \mathbb{N}$. *If* A *and* B *are two multisets over* X*:* $A, B \in \mathsf{Multiset}(X)$*, then* $A \uplus B$ *is the multiset over* X *defined by:* $(A \uplus B) : x \mapsto A(x) + B(x)$ *for all* $x \in X$.

$\emptyset_X$ *is the empty multiset over* X *defined by:* $\emptyset_X : x \mapsto 0$ *for all* $x \in X$*, and if* $a \in X$*,* $\{a\}_X$ *is the function* $A \in \mathsf{Multiset}(X)$ *s.t.* $A(x) = 1$ *if* $x = a$ *and* $A(x) = 0$ *otherwise.*

The objects considered here are those over which Dominance Constraints are expressed: **finite constructor trees** (i.e. the trees mentioned in §2.2.1, defined in [Kol04, chap. 2]). They are thus trees τ equipped with a canonical tree structure $\mathcal{M}_\tau$ and decorated with **labels** (over a given signature Σ) and node addresses.

With the following definition, the formal stepwise, bottom-up[6] process of proof-checking is defined, ensuring that:

1. With F: The formulas obtained at each step actually correspond to introduction or elimination of linear implication;[7]

2. With ft, additionally: One keeps track of hypotheses all along the process.

Function ft emulates the structurally inductive verification of the derivation tree:

[5] In all empirical cases the occurrences of modifiers will be just free or bound, actually, but multisets must be kept, for the sake of generality.

[6] Trees are usually displayed with leaves down, but proofs (derivations) the other way around, so "bottom-up" refers to solution trees.

[7] Implication elimination demands that the formula currently obtained actually *be* an implication.

- The axiom part passes the corresponding formula and no hypothesis to keep track of;
- The hypothesis part also passes the corresponding formula but initialises the hypothesis counter to one;
- The implication elimination part verifies that unification is possible and performs it when possible, while it just combines the bags of hypotheses together (there comes the handiness of multisets at play);
- Finally, the implication introduction part just augments the formula with a premiss while it actually checks that hypotheses are sufficient for enabling the rule.

We allow the definition to be generally defined over any given set of formulas,[8] although it will for us turn out to be Core Glue without quantifiers.

Definition 13 (Tree-to-formula mappings) *Let $\mathcal{L}$ be a set of linear logic formulas. Let $\mathcal{T}$ be the set of* **finite constructor trees** *over the (possibly infinite) signature $\Sigma = \{(\multimap_e)^2\} \cup \{(\multimap_i)^1_\varphi; \varphi \in \mathcal{L}\} \cup \{(\mathrm{ax})^0_\varphi; \varphi \in \mathcal{L}\} \cup \{[\varphi]^0; \varphi \in \mathcal{L}\}$.*

Then ft *and* F *are the partial functions from $\mathcal{T}$ onto, respectively, $\mathcal{L} \times \mathsf{Multiset}(\mathcal{L})$ and $\mathcal{L}$, inductively defined by:*

- $\mathsf{F}((\mathrm{ax})_j) := \alpha$ *and* $\mathsf{ft}((\mathrm{ax})_j) := (\alpha, \emptyset)$ *if the j-th axiom $\mathcal{A}_j$ is formula α;*
- $\mathsf{F}([\varphi]) := \varphi$ *and* $\mathsf{ft}([\varphi]) := (\varphi, \{\varphi\})$*;*
- $\mathsf{F}((\multimap_e)(x, y)) := \chi'$ *if* $\mathsf{F}(x) = \varphi \multimap \chi$ *and* $\mathsf{F}(y) = \psi$*, and* $\mathsf{ft}((\multimap_e)(x, y)) := (\chi', A \uplus B)$ *if* $\mathsf{ft}(x) = (\psi \multimap \chi, A)$ *and* $\mathsf{ft}(y) = (\psi', B)$*; where* $\chi' := \mathsf{mgu}(\psi, \psi')(\chi)$ *and ψ subsumes ψ'—both* ft *and* F *are* undefined otherwise*;*

  That is, $\mathsf{ft}((\multimap_e)(x, y))$ *is defined if*

  $$\frac{\mathsf{F}(x) \quad \mathsf{F}(y)}{\mathsf{F}((\multimap_e)(x, y))}\ (\multimap_e)_u$$

  is a valid application of modus ponens modulo unification;
- $\mathsf{F}((\multimap_i)_\varphi(x)) := \varphi \multimap \mathsf{F}(x)$*; If* $\mathsf{ft}(x) = (\psi, \{\varphi\}_{\mathcal{L}} \uplus A)$*, then* $\mathsf{ft}((\multimap_i)_\varphi(x)) := (\varphi \multimap \psi, A)$*;*

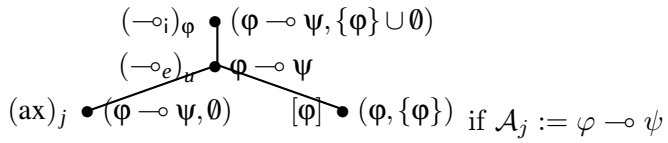
  That is, $\mathsf{ft}((\multimap_i)_\varphi(x))$ *is defined if*

  $$\frac{\begin{array}{c}[\varphi] \\ \vdots \\ \mathsf{F}(x)\end{array}}{\mathsf{F}((\multimap_i)_\varphi(x))}\ (\multimap_i)_\varphi$$

  is a valid implication introduction rule for linear logic.

The modelling of a proof by our new derivation trees is illustrated in figure 5.4. If both are defined, $\mathsf{F}(\tau)$ always is the first projection of the pair: $\mathsf{F}(\tau) = \mathsf{pr}_1(\mathsf{ft}(\tau))$.

[8] A sublogic (viz., a set of formulas) is usually referred to as a **fragment**, too. We will not use "fragment" in this latter sense, in order not to confuse with the sense pertaining to Dominance Constraints.



$$\dfrac{\dfrac{}{\varphi \multimap \psi}\,(\mathrm{ax})_j \qquad [\varphi]}{\dfrac{\psi}{\varphi \multimap \psi}\,(\multimap_i)_\varphi}\,(\multimap_e)_u$$

Figure 5.4: Behavior of the tree-to-formula mappings ft and F. Modeling of the proof (right) is achieved by bottom-up recursion (left): The nodes are displayed with their labels on the left and their images under ft on the right.

Remark. Interestingly, both last inductive cases are opposed as for their behavior w.r.t. LL formula checking and hypothesis book-keeping, respectively: The implication elimination $(\multimap_e)_u$ does not put any constraint on hypotheses but demands the formula $\mathsf{F}(x)$ be an implication, and $\mathsf{F}(y)$ be unifiable with its premiss; Conversely, the implication introduction $(\multimap_i)_\varphi$ augments *any* formula with premiss φ (no LL-structural constraint), but demands an according hypothesis $[\varphi]$ be present. We will notice how important this is later in the technicalities of the proof: It points at the difference between a partial correct and a correct derivation tree (defined below).

At this point we want to express how the machinery just introduced corresponds to our intuitive notion of a correct derivation tree: For the proof of φ to be correct, all hypotheses must have been discharged at its root.

Definition 14 (Correct derivation) *A* **correct derivation tree** *for Glue formulas is a finite constructor tree τ over Σ such that:*

$$\mathsf{ft}(\mathcal{R}(\tau)) = (\varphi, \emptyset_{\mathcal{L}}) \text{ for some formula } \varphi$$

For instance, the derivation tree shown in figure 5.4 is not correct, because hypothesis φ is still pending at the root. It is a partial correct tree, though, since all nodes correspond to valid rules (definition 16 below).

Furthermore, a crucial condition on Glue derivations is that *all* axioms must be used and *only once* (cf. §3.1).

Definition 15 (Correct derivation from axioms) *Let $\mathcal{A} = \mathcal{A}_1, \ldots, \mathcal{A}_n$ be a sequence of Glue axioms, φ a Glue formula. A* **correct derivation tree for φ from $\mathcal{A}$** *is a correct derivation tree τ with $\mathsf{ft}(\mathcal{R}(\tau)) = \varphi$ and such that the multiset of all axiom nodes $(\mathrm{ax})_j$ from τ contains* exactly *the elements $\mathcal{A}_1, \ldots, \mathcal{A}_n$.*

The following definitions introduce *incomplete* derivation trees, that will be useful in the forthcoming details of the proof, as a transition toward a fully correct tree: A correct derivation tree will be a partial correct derivation tree whose root has no hypothesis pending.

Definition 16 (*Partial* correct derivation tree) *A* **partial correct derivation tree** *is a tree τ for which mapping ft from definition 13 is well-defined.*

That is, the set $\mathcal{T}' \subseteq \mathcal{T}$ of partial correct derivation trees is the greatest subset of $\mathcal{T}$ that makes ft a total function.

The weak version of the definition above is meant to ensure the same conditions *up to hypothesis book-keeping.*

Definition 17 (Partial *weakly* correct derivation tree) *A tree $\tau \in \mathcal{T}$ is a* **partial weakly correct derivation tree** *if $\mathsf{F}(\tau)$ is well-defined.*

Remark. Obviously, if $\mathsf{ft}(\tau)$ is well-defined, so is $\mathsf{F}(\tau)$. In other words, any partial correct derivation tree is a partial weakly correct derivation tree. The latter type of tree merely checks the formulas passed upward have the right form, while the former additionally checks whether hypotheses are properly updated.

5.3 The translation itself

Having formalised the notions of correct derivation trees, we have set the stage for our translation: a Glue **axiom sequence** $\mathcal{A} = \mathcal{A}_1, \ldots, \mathcal{A}_n$ as input and a **dominance constraint** as output (as shown in figure 5.1), whose solutions will be proved correct derivation trees.

The translation itself is decomposed in two distinct steps, accordingly to the segregation of constraint atoms (or dominance graph edges) into **labelling constraints** (resp. tree edges) and proper **dominance constraints** (resp. dominance edges); Thus, properties of the translation will also fall into two categories: those which hold for the mere fragments (which correspond to labelling constraints): §5.3, and those which apply to the global output constraint: §5.4.

1. First, the **fragments** are obtained (definition 19): §5.3.1. Their definition extracts the logical structure of the axioms to rebuild it as a (piece of) derivation tree, with distinct treatments according to whether logical resources correspond between axioms of the global axiom sequence $\mathcal{A}$. The criterion is whether premisses of an axiom $\mathcal{A}_i$ match the suffix of another axiom $\mathcal{A}_j$ (in a sense to be defined below), and the goal is to introduce dependencies "hypothesis ($[\varphi]$) vs. binding ($(-\circ_i)_\varphi$)" in the output derivation.

   Such a definition induces an intuitive induction scheme: left-leaf-root induction (proposition 4), whose structure clings to definition 19. It is used to prove the numerous properties which can already be stated from fragments alone (constraint $\mathsf{fragm}(\mathcal{A})$): §5.3.2.

   The most important of these are stated in §5.3.3, concluding the fragment part: proposition 10 provides a *restricted* correctness of the output fragments (weak local soundness), while lemma 11 makes the relations between fragments explicit.

2. Then **dominance** is expressed between fragments (definition 25). This presupposes a classification of fragments according to their logical content: definition 23.

   The resulting constraint $\mathsf{T}(\mathcal{A})$ is proved to be well-formed w.r.t. Dominance Constraints: normality (proposition 13) and leaf-labeledness are obtained. In particular, they justify the use of dominance *graphs* in place of constraints.

In §5.4.2, the connection is made to the polarity of Implicative Linear Logic formulas (theorem 1), and a strengthened version is used as a crucial assumption (a polarity pattern (1) tailored for Glue, taken over from [GL98]) to restrict the polarities in $\mathsf{fragm}(\mathcal{A})$ and therefore to further constrain $\mathsf{T}(\mathcal{A})$ (corollaries 15 and 16).

In §5.4.3, finally, soundness is gradually proved: via partial weak soundness (proposition 18) and partial soundness (proposition 19), i.e. firstly without taking hypothesis book-keeping into account, and secondly doing it, but overlooking superfluous hypotheses; Thus, the final result (theorem 20) just says that all hypotheses must have been consumed when the root of a solution derivation tree is reached.

5.3.1 Translation—fragments

Implicative formulas Here a further structural definition on formulas is needed in the theoretical apparatus, additionally to those from §3.5.2.

Definition 18 (Non-ψ-modifying axioms) *If $\mathcal{L}$ is a set of formulas, ψ a formula, let $\mathcal{N}_\psi(\mathcal{L})$ denote $\{\varphi \in \mathcal{L};\ \psi \notin \mathsf{Suf}(\varphi) \cap \mathsf{Prm}_r(\varphi)\}$, that is, the set of formulas which ψ is not both a suffix and a right-branching premiss of.*

The "non-modifying" restriction to definition 19 below proves useful in §5.4.2.

Computing fragments. The fragment translation of a Glue axiom set $\mathcal{A}$ will firstly extract the tree structure of all LL axioms and translate it in the constraint language: rule names as labels and branching numbers of these rules as label arities (that is: 0 for an axiom introduction, 1 for a implication introduction and 2 for an implication elimination).

We denote by $1^{\mathsf{depth}(\varphi)}$ a word over the alphabet $\{1,2\}$ constituted of as many 1s as the (implicational) depth of the axiom φ.[9] It is thus the address of node $(\mathrm{ax})_j$ in fragment $\mathsf{fragm}(\mathcal{A}_j)$.

Definition 19 (Translation—labelling constraints) *If φ is the j-th axiom in $\mathcal{A}$ ($\varphi = \mathcal{A}_j$), then*

$$\mathsf{fragm}(\varphi) := X_j^s : (\mathrm{ax})_j \ \wedge\ z_j^s(\varphi), \text{ where } s = 1^{\mathsf{depth}(\varphi)} \text{ and} \tag{5.2}$$

$$z_j^\varepsilon(a) := \top \text{ if } a \text{ is an atom of type } t \text{ or a variable;} \tag{5.3}$$

$$z_j^{u1}(\psi \multimap \chi) := \begin{cases} X_j^u : (\multimap_e)(X_j^{u1}, X_j^{u2}) \ \wedge\ z_j^u(\chi) \ \wedge\ X_j^{u2} : [\psi] & (5.4) \\ \qquad \text{if } \psi \text{ does not subsume any } \psi' \in \mathsf{Suf}(\mathcal{N}_\psi(\mathcal{A} \backslash \mathcal{A}_j)) & \\ X_j^u : (\multimap_e)(X_j^{u1}, X_j^{u2}) \ \wedge\ z_j^u(\chi) \ \wedge\ w^{u2}(\psi') & (5.5) \\ \qquad \text{if } \psi \text{ subsumes some } \psi' \in \mathsf{Suf}(\mathcal{N}_\psi(\mathcal{A} \backslash \mathcal{A}_j)) & \end{cases}$$

with

$$w^u(\psi \multimap \chi) := X_j^u : (\multimap_i)_\psi(X_j^{u1}) \ \wedge\ w^{u1}(\chi) \tag{5.6}$$

$$\text{and } w^u(a) := \top \text{ if } a \text{ is an atom of type } t \text{ or a variable.} \tag{5.7}$$

[9] If the logic $\mathcal{L}$ is purely implicative, the **(implicational) depth** of a formula $\varphi \in \mathcal{L}$ is $\mathsf{depth}(\varphi) := 0$ if φ is an atom or a variable, and $\mathsf{depth}(\chi) + 1$ if $\varphi = \psi \multimap \chi$. It will also be abbreviated in δ when the context is clear.

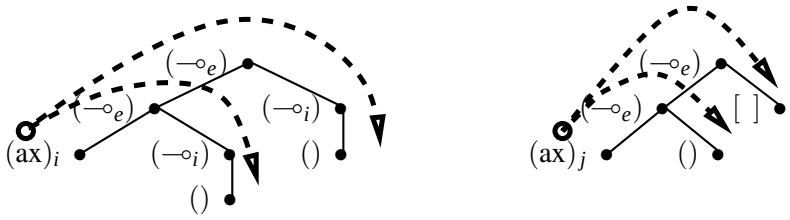


Figure 5.5: Inductive definition of fragments: path.

The definition shows how the tree structure implicit in each formula $\mathcal{A}_j$ is inductively reconstructed. Yet it does not start from the root, but from the axiom node, along the scheme suggested in figure 5.5, which shows the form of the output fragments.. This path is directed "from left to leaf (downward) and root (upward)".

Remark. Both indices involved in the notations (for z and X) are necessary: One needs to keep track of the axiom number, so as not to confound the X_js with some X_is from another fragment, but above all of the *current position* in the tree so far (the **addresses** of nodes in a finite constructor tree): At the beginning, position is set at the position the axiom node is known to occupy—the axiom's depth (5.2), only to be decremented upward (lines (5.4) and (5.5)) or incremented downward (line (5.6)). Addressing stops upon encountering a leaf (5.7) or the root (5.3).

Example 1 *Let us see on the example of two actual Glue entries how the fragment is built step by step; This is illustrated in figure 5.6. Entries are taken from the* EMBASSY *axiom set (simplified QNP form), standing for* Every man believes that a student yawns *(cf. figure 3.11, §3.6):*

$$\begin{array}{lll} [\textbf{every-man}] & : \mathcal{A}'_1 = \forall M\ (\overline{m \multimap M}) \multimap M \\ [\textbf{believes}] & : \mathcal{A}_3 = \underline{y} \multimap \overline{m \multimap f} \\ [\textbf{a-student}] & : \mathcal{A}'_4 = \overline{\forall S\ (s \multimap S)} \multimap S \\ [\textbf{yawns}] & : \mathcal{A}_6 = s \multimap \underline{y} \end{array}$$

1. *QNP:* $\mathcal{A}'_1 = (m \multimap M) \multimap M$

   Firstly let us recall that we consider only quantifier-free formulas, i.e. function fragm *will actually process* $(m \multimap M) \multimap M$ *Secondly, in a well-formed Glue axiom sequence, the QNP is known to match its verb. This is expressed as modified scope resource* $m \multimap M$ *matching (subsuming) the suffix* $m \multimap f$ *of the verb's axiom* $y \multimap m \multimap f$*. Therefore:*

$$\begin{array}{rcl} z_1^{11}((m \multimap M) \multimap M) & := & X : (\multimap_e)(X_\delta, Z)\ \wedge\ z_1^1(m \multimap M) \\ \quad \textit{with}\ z_1^1(m \multimap M) & = & Y : (\multimap_e)(X, \Theta) \wedge w^2(m \multimap M)\ \wedge\ \underbrace{z_1^\varepsilon(M)}_{\varepsilon} \end{array}$$

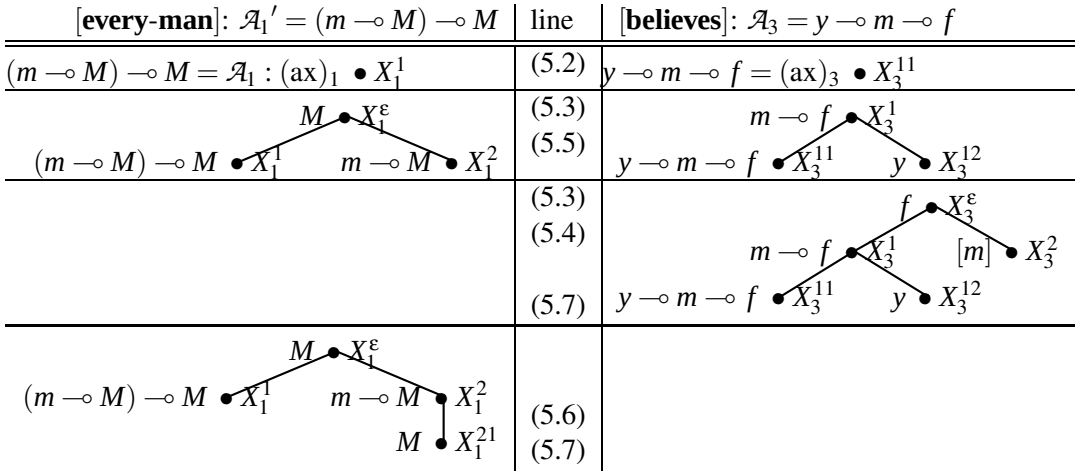


Figure 5.6: Piecemeal fragment construction from axioms: a quantified noun phrase and a sentence-embedding verb.
The formulas passed up to z_j^u (resp. the variables involved) are displayed on the left (resp. on the right) of each node.

$$\textit{where } w^2(m \multimap M) \quad = \quad Z : (\multimap_i)_m(Z') \ \wedge \ \underbrace{w^{21}(M)}_{\varepsilon}$$

2. *Sentence-embedding verb:* $\mathcal{A}_3 = y \multimap m \multimap f$

   This class of verbs has the particularity in Glue that the final resource y for the embedded sentence is matched: y is also the suffix of the axiom $\mathcal{A}_6$, while the resource s for the subject does not (only quantified in $\mathcal{A}_1'$). This results in differentiated treatment of these two premisses y and m: one as a hole (y) and the other one as a hypothesis $[m]$.

$$z_3^{11}(y \multimap m \multimap f) \quad := \quad X : (\multimap_e)(X_\delta, Z) \ \wedge \ w(y) \ \wedge \ (Y : (\multimap_e)(X, \Theta : [m]) \ \wedge \ (\underbrace{z_3^{\varepsilon}(f)}_{\top}))$$

Proposition 2 *Under assumption 7, the function* fragm *of Glue axioms from definition 19 is* totally *defined.*

Proof. It is enough to show that $z_j^s(\varphi)$ is defined for any formula φ whose implicational depth is $|s|$. Structural induction on φ:

- base case: if φ is an atom of type t or a variable, then its implicational depth is 0, so that $s = \varepsilon$ and line 5.3 applies and stops the recursion.
- otherwise: $\varphi = \psi \multimap \chi$ (lines (5.4) and (5.5))

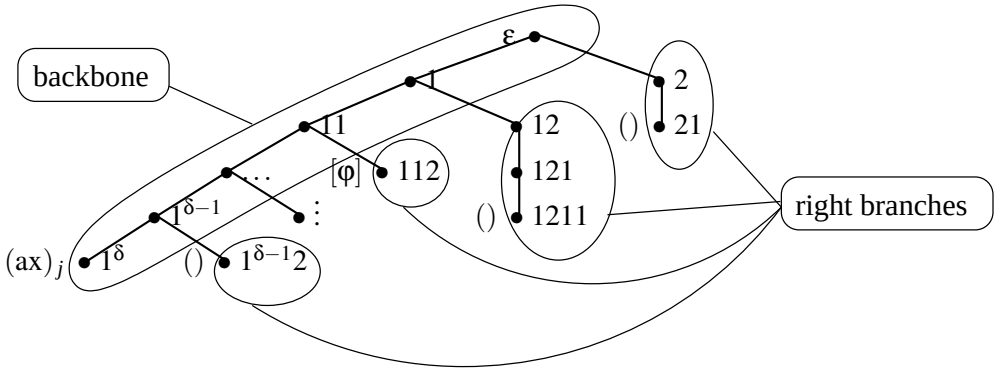


Figure 5.7: The comb structure of output fragments: the backbone and the dents (right-branch variables), which lead either to hypotheses: $[\varphi]$, or to unlabelled variables (viz. holes): ().

(5.4) if ψ does not subsume any element in $\mathsf{Suf}(\mathcal{A})$, then $z_j^{u1}(\psi \multimap \chi)$ is defined, because so is $z_j^u(\chi)$ (IH) and $\mathsf{depth}(\chi) = \mathsf{depth}(\psi \multimap \chi) - 1$;

(5.5) if ψ subsumes some $\psi' \in \mathsf{Suf}(\mathcal{A})$, then this time $z_j^{u1}(\psi \multimap \chi)$ is defined iff $w^{u2}(\psi')$ is also defined. By induction (lines (5.6) and (5.7)), one sees that, for any u, $w^u(\psi')$ is defined iff $\mathsf{cc}^+(\psi')$ is an atom of type t or a variable. But since $\psi' \in \mathsf{Suf}(\mathcal{A})$, $\mathsf{cc}^+(\psi')$ is the conclusion of a formula of $\mathcal{A}$, and thus, by assumption 7, of the required form.

□

Definition 20 (Resulting fragments) *If $\varphi = \mathcal{A}_j$ is an axiom, then $F_j := \mathsf{fragm}(\varphi)$ is the* **fragment** *associated to it.*

Definition 21 (Variable types) *Variables of the form X_j^u with $u \in 1^*$ are called* **backbone** *variables; Those of the form X_j^u with $u \in 1^*21^*$ are called* **right-branch** *variables.*

Dominance atoms where a backbone (resp. right-branch) variable occurs are called backbone (resp. right-branch) atoms.

The following lemma shows that all variables of a fragment are actually of either one type. We see that the choice of the variable exponents in definition 19 was not casual: These happen to be exactly the **addresses** of nodes in their respective fragments. Thus, these addresses are as restricted as their variables, according to the inductive cases of definition 19:

Lemma 3 (Atom addresses) *For every j and every $X_j^u \in \mathcal{V}(A_j)$, $u \in 1^*|1^*21^*$, i.e. there may be at most one 2 in any variable's address.*

Proof. In definition 19, the second case (5.4 and 5.5) is the only one introducing a 2 (in $u2$). It cannot be applied twice, since (5.4) yields a recursive call over a strictly smaller 2-free word (u); and so does (5.5), plus a function (w^{u2}) that does not yield any other 2 (induction cannot escape (5.6) and (5.7)). □

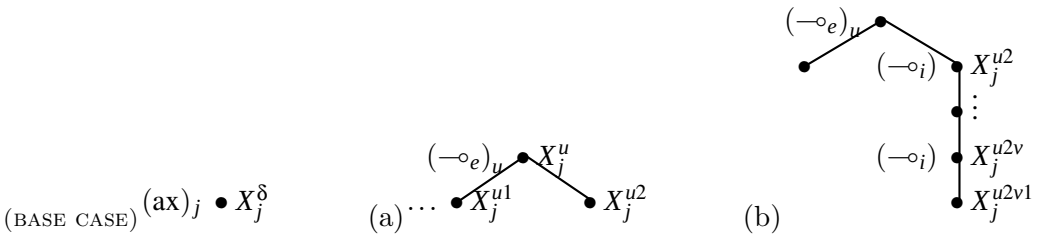


Figure 5.8: The left-leaf-root induction along fragments: two induction cases.

Remark. As for the structure of the fragments, the lemma implies that right-branching occurs at most once in any root-leaf path of the fragments. Hence, all output fragments are comb-shaped, where dents of the comb lead either to an hypothesis (short) or to a hole (short or long): cf. figure 5.7.

5.3.2 Properties of the labelling constraints

An induction tailored for fragm**.** In the subsequent properties and proofs, a lot of induction will be used on the outputs of function fragm from definition 19. Basically, all proofs are alike, or more precisely proceed of an induction along the same path in the outcoming tree, as illustrated in figure 5.8.

Therefore, we design our own induction scheme to help clarify the proofs following. It is defined and proved a valid induction scheme (in terms to be defined below), and then it will be easily and visually checked that the properties to be proved comply to it.

Proposition 4 (Left-leaf-root induction schema) *Let $\mathcal{P}$ be a property of constraint variables and F_j a fragment. If $\mathcal{P}$ verifies:*

1. (base case) *$\mathcal{P}(X_j^\delta)$ (true for the axiom node);*
2. (induction case)
   (a) *if $\mathcal{P}$ holds for a non-root backbone variable (i.e. for a X_j^{u1} with $u \in 1^*$), then it holds for its* parent *X_j^u and for its (only)* sibling *X_j^{u2}:*

$$(\mathcal{P}(X_j^{u1}) \wedge u \in 1^*) \Longrightarrow (\mathcal{P}(X_j^u) \wedge \mathcal{P}(X_j^{u2}));$$

   (b) *if $\mathcal{P}$ holds for a right-branch variable (i.e. for a X_j^{u2v} with $u, v \in 1^*$), then it holds for its child X_j^{u2v1},* if it exists*:*

$$\mathcal{P}(X_j^{u2v}) \stackrel{!}{\Longrightarrow} \mathcal{P}(X_j^{u2v1});$$

then $\mathcal{P}$ holds for all variables of fragment F_j.

Now we also have to prove that this is indeed a proper induction scheme w.r.t. the fragments obtained on output of definition 19, that is, that all variables $X = X_j^u$ of any fragment are hit by the induction.

Proof. Let us define $\mathcal{I}$ as the smallest set of variables of fragm$(\mathcal{A}_j)$

- containing $\mathcal{I}_0 := \{X_j^{1^{\mathsf{depth}(\mathcal{A}_j)}}\}$ and
- closed under
  (a) parent and sibling relations for $u \in 1^*$:
  $$X_j^{u1} \in \mathcal{I} \wedge u \in 1^* \Longrightarrow X_j^u, X_j^{u2} \in \mathcal{I}$$
  (b) child relation for $u \in 1^*21^*$:
  $$X_j^{u2v} \in \mathcal{I} \wedge X_j^{u2v1} \in \mathcal{V}(\mathsf{fragm}(\mathcal{A})) \Longrightarrow X_j^{u2v1} \in \mathcal{I}$$

Now, to prove remains only:

Lemma 5 *$\{X;\ X \in \mathcal{V}(F_j)\} \subseteq \mathcal{I}$, that is, all nodes of fragment j are touched by the span set $\mathcal{I}$ of left-leaf-root induction.*

The proof consists of two separate inductions, respectively:

1. for all backbone atoms, i.e. variables $X = X_j^u$ s.t. $u \in 1^*$: induction on $n = \delta - |u|$:
   - $n = 0$
     $u = 1^\delta$: this is also the base case of left-leaf-root induction and means that $X \in \mathcal{I}_0 \subseteq \mathcal{I}$;
   - $n \rightsquigarrow n + 1$
     Let u' be such that $n + 1 = \delta - |u'|$; this u' with $|u'| := \delta - n - 1$ then verifies $u = u'1$ (and in particular $u' \in 1^*$) and matches (5.4) from definition 19. Therefore, $X_j^{u'}$ is *parent* of X_j^u and, since $X_j^u \in \mathcal{I}$ (induction hypothesis), $X_j^{u'} \in \mathcal{I}$ (definition of $\mathcal{I}$, (a));
2. for all right-branch variables, X_j^u s.t. $u = v2w$ $(v, w \in 1^*)$: induction on $n = |w|$:
   - $n = 0$
     $w = \varepsilon$, so that $u = v2$ with $v \in 1^*$.
     The first point of this very proof tells us that $X_j^t \in \mathcal{I}$ for all $t \in 1^*$, in particular for $t = v1$. Case (a) of definition of $\mathcal{I}$ says then that it also holds for the *sibling* $X_j^{v2} = X_j^u$;
   - $n \rightsquigarrow n + 1$
     Suppose the property true for $u = v2w$ with $|w| = n$, then consider $u' := u1 = v2w1$ (only possibility since there is at most one 2 in each address); by case (b) of definition of $\mathcal{I}$ and induction hypothesis, $X_j^{u1} \in \mathcal{I}$.

By lemma 3, the two inductions cover all possible cases. □

Remark. The proof just exposed is the core and the justification of the whole induction scheme. It would be therefore strictly equivalent, albeit less intuitive, to prove the forthcoming properties by means of the two separate inductions from above.

Unicity of variables. The next proposition constitutes a preliminary step toward a correct output, by giving a first well-formedness property of output fragments: All variables issued from fragm uniquely label the output constraint, if at all.

Proposition 6 (First well-formedness property) *Any variable $X \in \mathcal{V}(F_j)$ appears at most once on the left-hand side of a labelling atom in F_j.*

Since different fragments are also distinctively indexed, the property will extend to the greater constraint $\mathsf{fragm}(\mathcal{A}) := \bigwedge_j F_j$.

Proof. The left-leaf-root induction proof that any $X = X_j^u$ appears at most once on the left is merely a rereading of definition 19:

Every recursive call via z_j^u makes the address u borne by labelling variables strictly smaller (backbone: (5.4) and (5.5)). Every recursive call via w_j^u makes addresses strictly longer, but with a unique 2 (right branches: (5.6)).

Besides: (BASE CASE) The axiom node $X = X_j^u$ with $u = 1^{\mathsf{depth}(\mathcal{A}_j)}$ obviously appears only in the first atom $X_j^\delta : (\mathrm{ax})_j$. Hence, all variables uniquely introduce labels. □

"Type" inference. What the translation outputs so far for each axiom, via fragm, is a tree labelled with LL rule names: a piece of derivation tree, as defined in §5.2. But only the leaves (up to those that are holes) give information about the actual *formula* meant to appear (as an intermediate conclusion of the derivation tree).

It probably appeals more to the intuition to give trees labelled with the formulas, which are piecemeal derived with every derivation step performed. For instance, in figure 5.9 one would like to be able to infer, from the tree on the left, the tree on the right, which directly shows that W is derived from $(v \multimap r) \multimap (w \multimap W) \multimap W$ (provided proofs for W and r are already present).

In fact, the following definition introduces **type inference**[10], a mapping that can infer the formula (type) expected at any node, from the mere hypotheses and axiom node. In particular, type inference is able to return the formula expected at the holes, too; It thus happens to be a necessary criterion for linking fragments together (in terms to be defined below, §5.3.4): formulas for holes may not be apparent otherwise.

Type inference starts from the axiom formula and for each $(\multimap_e)$-node infers formulas for the parent (conclusion of the implication) and the sibling (premiss) nodes; for each $(\multimap_i)$-node it simply infers the child node (conclusion); That is, its computing follows exactly the left-leaf-root path introduced in the induction scheme above (proposition 4).

Definition 22 (Type inference) *We define the following procedure constructing subformulas of axioms from the variables $\mathsf{ti} : \bigcup_j \mathcal{V}(F_j) \to \mathcal{L}$:*

1. $\mathsf{ti}(X_j^s) := \mathcal{A}_j$ *if $s = 1^{\mathsf{depth}(\mathcal{A}_j)}$ (axiom node);*

[10] Type inference is actually a concept of computer science, one of the numerous interesting parallelisms the Curry-Howard Correspondence offers (cf. §3.3.2): The structure of λ-terms may similarly be retrieved from the formulas labelling the leaves (their **types**) and the rule names decorating the type-checking tree.

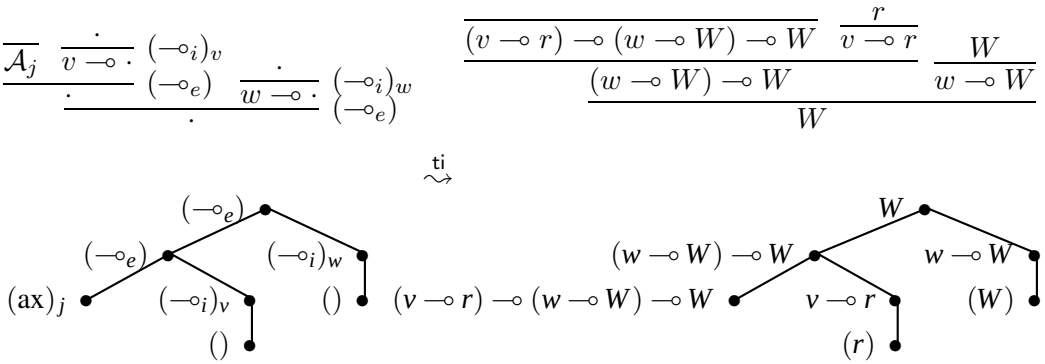


Figure 5.9: Type inference—the actual output for the determiner axiom [**every**]: $\mathcal{A}_j := (v \multimap r) \multimap (w \multimap W) \multimap W$ induces an injective mapping onto (occurrences of) subformulas of $\mathcal{A}_j$.

2. $\mathsf{ti}(X_j^u) := \psi$ *and* $\mathsf{ti}(X_j^{u2}) := \varphi$ *if* $\mathsf{ti}(X_j^{u1}) = \varphi \multimap \psi$ *with* $u \in 1^*$ *(backbone);*

3. $\mathsf{ti}(X_j^{u1}) := \psi$ *if* $\mathsf{ti}(X_j^u) = \varphi \multimap \psi$ *with* $u \in 1^*21^*$ *(right branch).*

The procedure, given as such, is not genuinely functional, we therefore prove in the next proposition it is indeed a function. Furthermore, the proposition provides a small "tool-box" that ensures that everything behaves as expected: ti returns the subformulas of each axiom decomposed in definition 19 (2.), and thereby returns the right LL-structure (3.).

Proposition 7 (Properties of type inference)

1. *Thus defined,* $\mathsf{ti} : \mathcal{V}(\mathcal{A}) \to \mathcal{L}$ *is a function defined for any fragments stemming from* $\mathcal{A}_1, \ldots, \mathcal{A}_n \in \mathcal{L}$*. That is, for any* $X_j^t \in \mathcal{V}(F_j)$*,* $\mathsf{ti}(X_j^t)$ *is uniquely defined.*

2. *It is such that every recursive call of a* z_j^u *in definition 19 is performed with* $\mathsf{ti}(X_j^u)$ *as argument. In other words: for all* j*,* u *and* φ*, if* $z_j^u(\varphi)$ *appears in* $\mathsf{fragm}(\mathcal{A}_j)$*, then* $\varphi = \mathsf{ti}(X_j^u)$*.*

   Likewise, if $w^u(\varphi)$ *appears in* $\mathsf{fragm}(\mathcal{A}_j)$*, then* $\varphi = \mathsf{ti}(X_j^u)$*.*

3. *It furthermore verifies: For all* $X, Y, Z \in \mathcal{V}(F_j)$*,*

   (a) *if* $X : (\mathrm{ax})_j$ *appears in* $\mathsf{fragm}(\mathcal{A})$*, then* $\mathsf{ti}(X) = \mathcal{A}_j$*;*

   (b) *if* $X : [\varphi]$ *appears in* $\mathsf{fragm}(\mathcal{A})$*, then* $\mathsf{ti}(X) = \varphi$*;*

   (c) *if* $X : (\multimap_e)(Y, Z)$ *appears in* $\mathsf{fragm}(\mathcal{A})$*, then* $\mathsf{ti}(Y) = \mathsf{ti}(Z) \multimap \mathsf{ti}(X)$

   (d) *if* $X : (\multimap_i)_\varphi(Y)$ *appears in* $\mathsf{fragm}(\mathcal{A})$*,then* $\mathsf{ti}(X) = \varphi \multimap \mathsf{ti}(Y)$*.*

Proof. All three results are easily obtained via left-leaf-root induction, since definition of ti follows the same left-leaf-root path. □

Characterisation of variables

The following two lemmas provide a summary of how variable addresses relate to, respectively, their position in the dominance graph (fragment, actually), and via type inference their LL-structural properties.

Lemma 8 *Let F_j be a fragment. For any variable $X = X^u_j$ appearing in $\mathcal{V}(F_j)$, X is*

1. *the* **root** *of F_j iff $u = \varepsilon$;*
2. *a* **hole** *of F_j iff $u \in 1^*21^*$ and $X^{u1}_j \notin \mathcal{V}(F_j)$;*
3. *labelled with $(\multimap_i)$ iff $u \in 1^*21^*$ and $X^{u1}_j \in \mathcal{V}(F_j)$;*
4. *a* **hypothesis** *of F_j iff $u \in 1^*2$ and $\mathsf{ti}(X)$ subsumes some suffix of a non-$\mathsf{ti}(X)$-modifying axiom.*

Proof. A simple rereading of definition 19 should convince that addresses fulfill this distribution.

In particular, with a little help from proposition 6 about unicity one sees that:

1. the only occurrence of X^ε_j appears when line (5.3) is called, that is, instantiated in $X^\varepsilon_j : (\multimap_e)(X^1_j, X^2_j) \wedge \top \wedge \cdots$; therefore, such a X^ε_j does not have any parent, and is thus the root (and conversely F_j's root may occur only in this line).
2. A hole of a dominance graph is a node with incoming but no outcoming tree edges, so a hole in the corresponding dominance constraint (cf. §2.2) is unlabelled but appears in one variable's label (at most one due to proposition 6). The only possibility for this to occur is in (5.7), which instantiates line (5.6) in $X^v_j : (\multimap_i)_\psi(X^{v1}_j) \wedge \top$; again, this is the only case where this labelling atom may occur, and X^{v11}_j obviously does not exist then.
3. Labels of the type $(\multimap_i)$ obviously occur only in (5.6) are exactly those variables that occur in (5.6) and are not covered by the case above; the same holds for variable addresses, hence the equivalence.
4. A hypothesis label occurs in (5.4), *if* the subsumption condition is verified, and only then; obviously, these atoms exactly yield those addresses of the form 1^u2 for which the subsumption condition is verified: ⟨QED⟩.

□

Structurally distinguished subformulas of Glue axioms, as defined in §3.5.2, are also reflected in the output fragment, via ti: The backbone are mapped to suffixes and right branches to right-branching premisses.

Lemma 9 *Let $F_j = \mathsf{fragm}(\varphi)$ be a fragment. For any variable $X = X^u_j$,*

1. *$\mathsf{ti}(X) = \mathsf{cc}^+(\varphi)$ iff $u = \varepsilon$;*
2. *$\mathsf{ti}(X) \in \mathsf{Suf}(\varphi)$ iff $u \in 1^*$;*
3. *if furthermore X appears in F_j, then $\mathsf{ti}(X) \in \mathsf{Prm}_r(\varphi)$ iff $u \in 1^*21^*$.*

Proof. It is easy to see that any suffix of an axiom has an antecedent variable under ti, because definition 19 exhaustively decomposes conclusions, via z_j^u in (5.4) and (5.5); so that the whole backbone of a fragment bijectively corresponds to the axiom's suffixes.

1. From proposition 7(3c and 3a) we easily see that $\mathsf{ti}(X) = \mathsf{cc}^+(\varphi)$ iff X is the root of F_j;
2. Likewise, proposition 7(3c, 3a) together with definition 22(2) inductively yield the result;
3. The very same points yield the result for $u \in 1^*2$. For $u \in 1^*21^+$, proposition 7(3d) and definition 22(3) also inductively yield the needed remaining result. Notice w.r.t. this that the set $\mathsf{Prm}_r(\varphi)$ recursively branches to the *right* of the linear implication and thus only in the $\mathsf{ti}(Y)$ in prop. 7(3d), so that any X_j^u with such u appearing in F_j *is* in $\mathsf{Prm}_r(\varphi)$.

□

5.3.3 Properties of the labelling constraints—weak local soundness

A first brick in the wall. The following proposition is the first important result of this section and represents about as much theoretical content as can be extracted from mere fragments, the constraint $\mathsf{fragm}(\mathcal{A})$ obtained so far still lacking the actual dominance information. No verification is performed yet on introduction or elimination of hypotheses (premature, too, as long as dominance has not been introduced).

It states that, *up to subsumption*, if any, the formulas obtained from any fragment node either through type inference (that is, unknowingly of what formulas are supposed to instantiate the Glue variables mapped from *holes*—the other holes are mapped to type t atoms and thus wholly determined) or through F (that is, with subsumption performed at $(-\circ_e)$-labeling atoms, and instantiating these variables) are the same.

Proposition 10 (Weak *local* soundness) *Let F_j be a fragment and $\mathcal{M}_\tau$ a constructive solution thereof (viz. a tree structure with τ a derivation tree, which together with an assignment α verify $(\mathcal{M}_\tau, \alpha) \models F_j$). Suppose $F(\tau.\alpha(X))$ defined for all $X \in \mathcal{V}(F_j)$.*

If $\mathcal{A}_j$ contains a Glue variable S, then there is a substitution σ of S and an address $u \in 1^*$ such that for every $X \in \mathcal{V}(F_j)$:

$$^{11}\ \mathsf{F}(\tau.\alpha(X)) = \begin{cases} \sigma(\mathsf{ti}(X)) & \text{if } X = X_j^{u2v} \text{ with } v \in 1^* \\ \mathsf{ti}(X) & \text{otherwise} \end{cases}$$

If $\mathcal{A}_j$ does not contain any variable, then for every $X \in \mathcal{V}(F_j)$:

$$\mathsf{F}(\tau.\alpha(X)) = \mathsf{ti}(X).$$

[11] The **selection** $\tau.\nu$ of a tree τ and one of its nodes ν denotes the greatest subtree of τ rooted in ν.

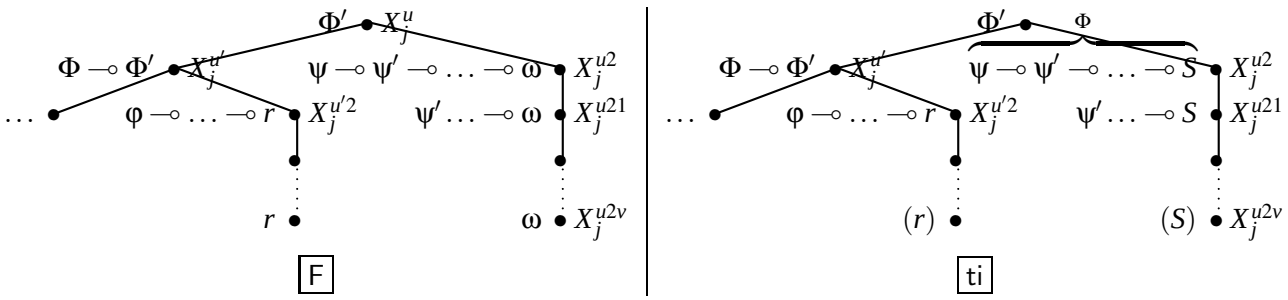


Figure 5.10: Weak local soundness.
F and ti coincide everywhere but on the rightmost right branch (starting at X_j^u): there substitution $S \mapsto \omega$ applies (yet only to the common conclusion S/ω of those formulas, because formulas $\psi, \psi', \ldots$ are variable-free (assumption 6)).
At the $(\multimap_e)$-node (X_j^u) substitution is performed for F (according to the definition of $(\multimap_e)_u$), so that both functions coincide on the whole backbone.
The example here would be an axiom F_j such that $\mathsf{Suf}(\mathcal{A}_j) \ni \Phi \multimap \Phi'$. [12]

That is, for any right branch of the tree but possibly one, F and ti coincide; for that other possible branch, they coincide *modulo subsumption*. This is illustrated in figure 5.10.

Proof. Let F_j be a fragment and $(\mathcal{M}_\tau, \alpha)$ a constructive solution of F_j.

In accordance with assumption 5, let S be the unique variable appearing in axiom $\mathcal{A}_j$, if there is one. So there is one unlabelled variable X_h in F_j. A solution implies that hole X_h is instantiated under F, too: $\mathsf{F}(\tau.\alpha(X_h)) := \omega$.

Let us then define a substitution, that is, an inductive mapping of formulas replacing variables with formulas: $\sigma : \mathcal{V}(\mathcal{L}) \to \mathcal{L}$:

$$\sigma(T) := \begin{cases} \omega & \text{if } T \text{ is variable } S \\ T & \text{if } T \text{ is a variable distinct from } S \end{cases} \tag{5.8}$$

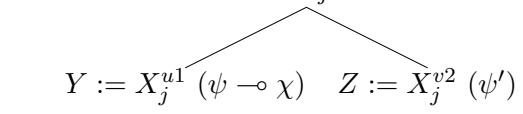
Using left-leaf-root induction for all $X \in \mathcal{V}(F_j)$:

- (BASE CASE) If $X = X_j^\delta := X_j^{\mathsf{depth}(\mathcal{A}_j)}$, definition 22 of type inference yields $\mathsf{ti}(X) = \mathcal{A}_j$.

  From the F side, since $X_j^\delta : (\mathrm{ax})_j$ appears in F_j and $(\mathcal{M}_\tau, \alpha) \models F_j$, we know that $\tau.\alpha(X_j^\delta) = (\mathrm{ax})_j$ and definition 17 of F then yields $\mathsf{F}(\tau.\alpha(X_j^\delta)) = \mathsf{F}((\mathrm{ax})_j) := \mathcal{A}_j$, the last being equal to $\mathsf{ti}(X_j^\delta)$: $\langle$QED$\rangle$.

- (INDUCTION CASE)

  (a) Suppose the property true for $Y = X_j^u$ with $u \in 1^+$, i.e. $u = v1$ for some $v \in 1^*$. Definition 19 uniquely (prop. 6) assigns labels to parent and sibling of Y the following way:



[12] Dotted edges just indicate here an ellipsis in the node structure, *not* dominance edges.

From definition 22, we get thus $\mathsf{ti}(Y) = \mathsf{ti}(Z) \multimap \mathsf{ti}(X)$; σ being a mere substitution, this also implies $\sigma(\mathsf{ti}(Y)) = \sigma(\mathsf{ti}(Z)) \multimap \sigma(\mathsf{ti}(X))$.

From the other side, since we have a **constructive solution** through τ and α, we know that $\tau.\alpha(X) = (\multimap_e)(\tau.\alpha(Y), \tau.\alpha(Z))$ and therefore, by definition of F and because $\mathsf{F}(\tau.\alpha(X))$ is defined:

$$\begin{array}{rcl} \mathsf{F}(\tau.\alpha(Y)) & = & \psi \multimap \chi \\ \mathsf{F}(\tau.\alpha(Z)) & = & \psi', \text{ subsumed by } \psi \ (\mathsf{mgu}(\psi, \psi')(\psi) = \psi') \\ \mathsf{F}(\tau.\alpha(X)) & = & \chi' := \mathsf{mgu}(\psi, \psi')(\chi) \end{array}$$

Starting from the Induction Hypothesis $\mathsf{F}(\tau.\alpha(Y)) = \mathsf{ti}(Y)$ one notes that $\mathsf{ti}(Z) = \psi$ and $\mathsf{ti}(X) = \chi$; to prove is:

i $\mathsf{F}(\tau.\alpha(X)) = \mathsf{ti}(X) = \chi$ and

ii $\mathsf{F}(\tau.\alpha(Z)) = \sigma(\mathsf{ti}(Z)) = \sigma(\psi)$.

It suffices to notice that $\mathsf{mgu}(\psi, \psi')$ and σ coincide on $\{\psi, \chi\}$ (because if there is a variable in ψ, it is S and $\omega = \mathsf{cc}^+(\psi')$), whence $\mathsf{F}(\tau.\alpha(X)) = \sigma(\psi) = \psi'$; likewise $\mathsf{F}(\tau.\alpha(Z)) = \chi = \chi'$.

(b) Suppose the property true for $Y = X_j^u$ with $u \in 1^*21^*$, i.e. $u = v2w$ for some $v, w \in 1^*$; let Y' be the child of Y, if it exists: $Y' := X_j^{u1} = X_j^{v2w1}$ (if it does not exist, induction trivially stops).

Then by definition 22, $\mathsf{ti}(Y') := \varphi \multimap \mathsf{ti}(Y)$ for some $\varphi \in \mathcal{L}$, and also $\sigma(\mathsf{ti}(Y')) := \sigma(\varphi) \multimap \sigma(\mathsf{ti}(Y))$. That is, by IH, $\sigma(\mathsf{ti}(Y')) = \sigma(\varphi) \multimap \mathsf{F}(Y)$.

But then **assumption 6** compels φ to be variable-free, and thus $\sigma(\varphi) = \varphi$, and $\sigma(\mathsf{ti}(Y')) = \varphi \multimap \mathsf{F}(Y)$, which by definition of F (def. 17) yields $\sigma(\mathsf{ti}(Y')) = \mathsf{F}(Y')$: $\langle$QED$\rangle$.

□

A glimpse at fragment relations. The lemma below is crucial, as will be seen further on in section 5.4, for linking different fragments with one another (unlike proposition 10 above, whose realm is a single fragment) and justifying dominance between them (to be introduced in §5.3.4). As such, it constitutes the most that the translation fragm can express about dominance—without having actually defined dominance yet; It also makes explicit the so far implicit dependencies among fragments needed in definition 19 (the difference between (5.4) and (5.5)).

Reusing the image of the comb structure of fragments (cf. figure 5.7), the lemma may be paraphrased as "type inference maps each dent of an axiom's fragment to a another axiom's backbone—modulo subsumption". Or: all right branches correspond each to an equally long (in terms of number of variables) part of another fragment's backbone. This is illustrated in figure 5.11.

Assumption 6 reveals here its purpose, in that it restricts the "modulo subsumption" proviso to the sole conclusion of an axiom (top node of the fragment).

Lemma 11 (Fragment linking) *For every hole $X \in \mathcal{V}(F_j)$, there exists a fragment F_i and a substitution σ of Glue variables such that $\sigma(\mathsf{ti}(X)) \in \mathsf{Suf}(\mathcal{A}_i)$.*

[13] Again, dotted edges indicate here only an ellipsis of fragment nodes, not dominance edges.

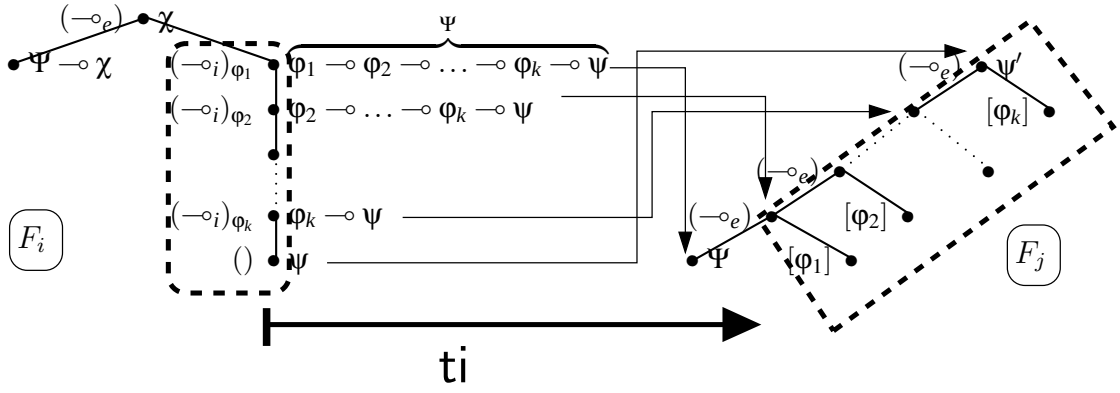


Figure 5.11: F_j's right branch must match the backbone of some F_i. The variables' labels are indicated on the left, their images under ti on the right.[13] There is matching *up to subsumption*: $\psi = \sigma(\psi')$ for some substitution σ.

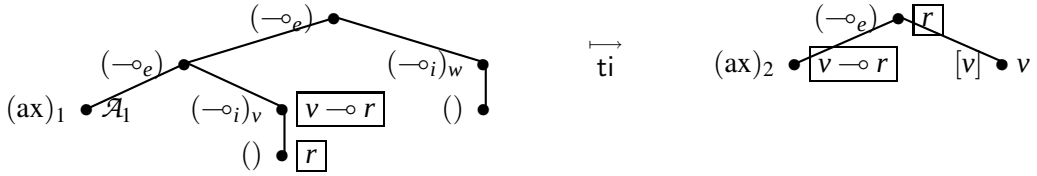


Figure 5.12: Fragment linking between the two entries for *every woman*.
Axiom $\mathcal{A}_1$ (*every*) depends on resource v from $\mathcal{A}_2$ (*woman*): fragment F_1 has a depth 2 right branch, which is thus to be found as a 2-node backbone part in fragment F_2.

In particular, for such a fragment, $\mathsf{ti}(\mathcal{R}(F_i)) = \mathsf{cc}^+(\sigma(\mathsf{ti}(X)))$ *for the same substitution* σ*.*

Proof. If $X = X_j^u$ is a hole, then (lemma 8) $u = vw$ with $v \in 1^*2$, $w \in 1^*$, so that the ancestor $Y = X_j^v$ of X verifies: $\mathsf{ti}(Y)$ subsumes some suffix ψ of $\mathcal{A}_i$ with $i \neq j$ (otherwise there would not be an X_j^{vw}: $z_j^v \models z_j^{vw}$): there is a substitution σ s.t. $\sigma(\mathsf{ti}(Y)) = \psi$.

We then have $\mathsf{ti}(X_i^{u'}) = \psi$ for some $u' \in 1^*$. But F_i's root is X_i^ε, s.t. $\mathsf{ti}(X_i^\varepsilon) = \mathsf{cc}^+(\mathcal{A}_i) = \mathsf{cc}^+(\psi) = \mathsf{cc}^+(\sigma(\mathsf{ti}(Y)))$ (since $\psi \in \mathsf{Suf}(\mathcal{A}_i)$). Under assumption 6, only suffixes of $\mathsf{ti}(Y)$ may contain variables, so that eventually at most its final conclusion is a variable and has a non-identical image under σ; hence $\sigma(\mathsf{ti}(Y)) = \sigma(\mathsf{cc}^+(\mathsf{ti}(Y)))$.

Besides, $\mathsf{ti}(X)$ happens to be this final conclusion $\mathsf{cc}^+(\mathsf{ti}(Y))$, which yields $\mathsf{ti}(\mathcal{R}(F_i)) = \mathsf{cc}^+(\sigma(\mathsf{cc}^+(\mathsf{ti}(Y)))) = \mathsf{cc}^+(\sigma(\mathsf{ti}(X)))$. □

For instance, the meaning entries a noun and its determiner (cf. §3.6) exhibit such a relation, as shown in figure 5.12.

A quick look back on section 5.3 shows what has been achieved so far: a definition of output fragments for each axiom, equipped with type inference, to make

their LL content always available; this content was shown to comply to any constructive solution of a fragment (weak local soundness), and to already give hints at what is left to construct (fragment linking).

5.3.4 Translation—proper dominance

The constraint obtained so far, possibly seen as the conjunction $\mathsf{fragm}(\mathcal{A}) := \bigwedge_{i=1}^{n} \mathsf{fragm}(\mathcal{A}_i)$ of all axioms' fragments, only give information about the *inner structure* of axioms, in a word: their semantic **argument structure**. A constructive solution of a fragment is, up to the nodes that fill the holes, still unique, yet ambiguity appears when multiple solutions are allowed, i.e. when constraints permit several plugging configurations (cf. Hole Semantics and MRS, chapter 4). This constitutes the key feature of *dominance* constraints: representing global **scopal structure**.

The next few steps will make explicit how our translation achieves it, and we will see that getting the global proper dominance constraint together requires a bit more designing than plain "plugging" of roots and holes.

In fact: not enough dominance atoms would mean less constraints on the output solution trees, and thus possibly more solutions than wanted, including spurious solutions; and so would misplaced dominance atoms. The design of dominance relations is therefore to be carefully examined.

We want our translation:

1. To "fill the holes" left open by fragm; the edges will thus have to connect these holes with adequate roots—the criterion of adequateness will be subsumption of both images under ti;

2. But also to express the relation between fragments exhibited in lemma 11 as genuine dominance: this relation stems from hypotheses made in fragments ($[\varphi]$) and used in others ($(-\circ_i)_\varphi$); the latter should thus be found *above* the former in a solution tree, hence a constraint on dominance.

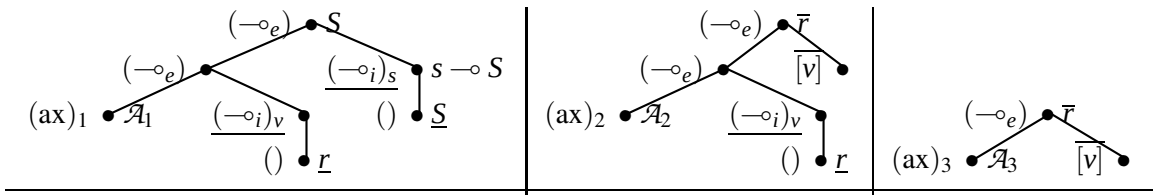
Accordingly, each output fragment will be classified as producing or consuming a semantic resource φ (a LL formula): cf. definition 23.

The relation between producers and consumers is both theoretically (theorem 1) and empirically (assumption 1) described. It relies on the polarity structure of Glue axioms (result inherited from linear proof theory, and the strengthened version thereof, an assumption tailored for Glue Semantics, first encountered in [GL98]). This will prove crucial in the final soundness steps (§5.4), because it formally justifies that the way dominance atoms are defined induces correct solutions to the output constraint.

Assign types to each fragment

The first step towards dominance between fragments is to identify and classify all fragments according to their behavior w.r.t. resources, that is, as to whether hypotheses introduction ($[\varphi]$) or binding ($(-\circ_i)_\varphi$) are present, and what kind of holes and roots equip the fragments:

Definition 23 *Let fragment F be a*



consumes	v, s	v	-
produces	-	v	v
(r, v)-consumer, (S, s)-consumer		(r, v)-replacer	(r, v)-producer

Figure 5.13: Fragment classification.

- (r, φ)**-consumer** *if it contains a hole X s.t.* $\mathsf{ti}(X) = r$ *with an ancestor of the same right branch*[14] *that is labelled $(-\circ_i)_\varphi$ and no $[\varphi]$-labelled node;*[15]
- (r, φ)**-producer** *if* $\mathsf{ti}(\mathcal{R}(F)) = r$ *and F contains a labelling atom $X : [\varphi]$, and is not a (g, φ)-replacer;*
- (r, φ)**-replacer** *if it contains both configurations that would make it both a (r, φ)-consumer and a (r, φ)-replacer, respectively:*
  - *a hole mapped to r and a right-branch label $(-\circ_i)_\varphi$ above.*
  - *a root mapped to r and a $[\varphi]$ label.*

In these cases r is an atomic Glue resource (LL atom or variable) (because of definition 19 type inference maps both holes and roots only to such formulas).

Remark. Note the definition stated here is a circular recursion, so that all three cases are mutually exclusive (that corresponds to how we would intuitively implement it).

Not all right branches comprise a $(-\circ_i)$-label, so that we will also classify fragments with depth-one right branches along the same scheme, with the dummy formula $\top$: Fragment F will be called a $(r, \top)$**-consumer** (resp., **-producer**, **-replacer**) if it fulfills definition 23 *up to the conditions on φ.*

Example 2 *The fragment for a sentence-embedding verb axiom (cf. believes in the* EMBASSY *sentence (3.11)) does no have any $(-\circ_i)$-label, and is thus a $(m, \top)$-consumer (as well as a (f, m)-producer): $\overline{y} -\circ \underline{m} -\circ f$ (cf. figure 5.5).*

The combinatorial constraint of polarity matching between fragments (§5.4.2) is so strong that it will often suffice to classify fragment after only one of their consumed (resp. produced) *hypotheses* (and root/hole checking will be superfluous):

Definition 24 *Fragment F is a* **consumer** (resp., **producer, replacer) for** φ *if there is a r s.t. F is a (r, φ)-consumer (resp. -producer, -replacer).*

[14] That is, if $X = X_j^{u2v}$ with $u \in 1^*$: a node $X_j^{u2v'}$ with $v' \leq v$ ("prefix of").

[15] In fact, such a hole $X := H_j^\varphi$ is known to be unique, not only within the right branch starting at X_j^u (due to the structure of definition 19), but also within the whole fragment F_j (yet only thanks to the polarity pattern—assumption 1).

Example 3 *Fragments thus classified may be assigned several types. For instance, the axioms for the quantified noun phrase* a sleepy student *yields fragments of all three types (cf. figure 5.13):*

$$\begin{array}{ll} [\mathbf{a}] & : \mathcal{A}_1 = (v \multimap r) \multimap (s \multimap S) \multimap S \\ [\mathbf{sleepy}] & : \mathcal{A}_2 = (v \multimap r) \multimap (v \multimap r) \\ [\mathbf{student}] & : \mathcal{A}_3 = v \multimap r \\ [\mathbf{yawns}] & : \mathcal{A}_4 = s \multimap y \end{array}$$

Yet the fragment for the determiner also is a consumer for the verbal resource s, but this relates to its relation to F_4 (which corresponds to its scopal behavior): F_1 is both a consumer for r and for s.

Adding constraints

Let us now informally expose which fragments may dominate which others: complementary hypothesis behavior (if any)—as $[\varphi]$ vs. $(\multimap_i)_\varphi$—induces dominance, but only if the hole and the root may actually be plugged (either equality of ti images, or subsumption):

1. Add a dominance constraint from every (r, Φ)-consumer to every (r, Φ)-replacer and every (r, Φ)-producer such that r' *subsumes* r;
2. Add a dominance constraint from every (r, Φ)-replacer (and every (r, Φ)-producer) to every (g, Φ)-consumer such that g' *subsumes* g;
3. No dominance constraint may ever join one fragment with itself; as it happens here, replacers may not dominate themselves;

where Φ may be $\top$ or an actual (variable-free) formula.

This is formally incorporated in the global dominance constraint as following:

Definition 25 (Translation—global) *Let $\mathcal{A}$ be a sequence of axioms. If $\mathsf{fragm}(\mathcal{A})$ is the conjunction $\bigwedge_j \mathsf{fragm}(\mathcal{A}_j)$, then $\mathsf{T}(\mathcal{A}) := \mathsf{fragm}(\mathcal{A}) \wedge \Delta$, where:*

$$\Delta := \bigwedge_r \bigwedge_{\Phi \in \{\top, \mathcal{L}\}} \left(\bigwedge_{F_k(r,\Phi)\text{-}consumer} \left(\bigwedge_{r' \text{ subsumes } r} \bigwedge_{F_j \ (r',\Phi)\text{-}producer \text{ or } \text{-}replacer} H_{k,g} \triangleleft^* \mathcal{R}(\mathcal{A}_j) \right) \wedge \bigwedge_{F_k(r',\Phi)\text{-}producer} \left(\bigwedge_{F_j \ (r,\Phi)\text{-}consumer \text{ or } \text{-}replacer} H_{k,g} \triangleleft^* \mathcal{R}(\mathcal{A}_j) \right) \right)$$

Remark. The above conjunction looks a priori computationally hazardous because of the distributed conjunctions, but it is all for the sake of rigor. The practical determination of constraints for a given axiom sequence $\mathcal{A}$ just proceeds by scanning all fragments and checking for other fragments matching, which amounts to the same, but yields a merely quadratic procedure in the number of fragments (that is, really tractable), as well as a quadratically expanded new constraint $\mathsf{T}(\mathcal{A})$.

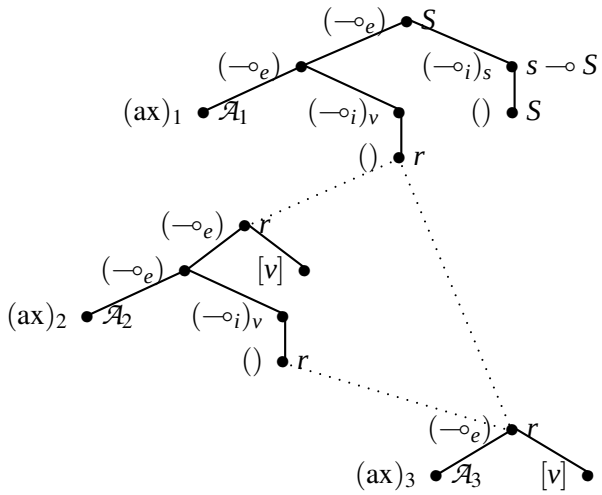


Figure 5.14: Dominances generated for *A sleepy student*.

The constraint $\mathsf{T}(\mathcal{A})$ being proved normal (next section), results from [Kol04, chap. 5] will almost directly provide the equivalence between dominance *graphs* and dominance *constraints*: modulo **compactification** of the constraint, $\mathsf{T}(\mathcal{A})$ has a solution iff its associated dominance graph has a solution.

And thus $\mathsf{T}(\mathcal{A})$ enjoys the advantages of being (equivalent to) a dominance graph: If the conjunction above were to produce any doubly defined dominance edges, they would just be skipped, because this does not make any difference for the graph. Likewise, the solving procedure sketched in §2.3.3 will eliminate redundant dominance edges/atoms, as are those between a producer and a consumer in presence of a replacer (cf., respectively, F_3, F_1 and F_2 in figure 5.14).

LL-structural consequence. The matching relation between a hole and a dominated root will be useful (cf. subsequent proofs) in the form of the following result.

Proposition 12 (Type inference for roots and holes) *If* $X \lhd^* Y$ *appears in* $\mathsf{T}(\mathcal{A})$*, then* $\mathsf{ti}(Y)$ *subsumes* $\mathsf{ti}(X)$*.*

Proof. This is evident from the definition of dominance atoms and from the fact that these solely connect holes to roots. □

5.4 Results

The following section offers, so to speak, the reward of the tortuous path we have just followed in the previous sections. Our theoretical artefact from section 5.3 meets here its purpose: a sound translation from Glue Semantics to Dominance Constraints, in section 5.4.3. The proof itself is decomposed in three subresults: Correctness of the output trees up to hypotheses book-keeping: **partial weak soundness** (18); Correctness up to the root node: **partial soundness** (19); Unconditional correctness of the output trees: **soundness** (20).

Connection is made to both the input and the output formalism, too: with dominance graphs in section 5.4.1 on normality as a well-formedness result, and with Glue Semantics in section 5.4.2 on polarity as a crucial proof argument in the last two results.

5.4.1 Normality

Here is where the first important connection is made between the two distinct formalisms—Glue and Dominance Constraints: Normality is a crucial well-formedness condition on dominance constraints for properties to apply such as equivalence with dominance graphs or the equivalence of solvability notions (cf. chain-connectedness, footnote 4).

Proposition 13 $\mathsf{T}(\mathcal{A})$ *is leaf-labeled and normal.*

Proof.

1. Leaf-labeledness amounts to: every unlabeled variable (hole) must be the head of a dominance atom.

   If X is a hole in F_j, then either F_j is a consumer for some resource ($X = X_j^{u21v}$, $u, v \in 1^*$) or F_j is a $(\mathsf{ti}(X), \top)$-consumer ($X = X_j^{u2}$). In either case, lemma 11 is applicable and exhibits a fragment F_i that is a producer or a replacer of the adequate type. There is then according to definition (25) a dominance atom $X \lhd^* \mathcal{R}(F_i)$: $\langle \textsc{qed} \rangle$.

2. Normality (definition 5, §2.2.3) demands a "checklist" for a given contraint C to be normal, consisting of the following four points:

   (a) (NO OVERLAP) This condition is obviously fulfilled because of how variable indices were chosen (in definition 19), namely, all different for all nodes; Besides, as evoked in §2.2.3, C may undergo $\cdot^{\neq}$-normalisation, which adds inequality atoms $X_j^i \neq X_{j'}^{i'}$ for all $(i, j) \neq (i', j')$.

   (b) (TREE-SHAPED FRAGMENTS) The well-formedness proposition 6 claims unicity of labelling heads as well as holes. The absence of cycles is granted by unicity of variable names and the structure of definition 19: neither dominance (different fragments) nor labelling atoms (different addresses) may branch to the same atom.

   (c) (DOMINANCES OUT OF HOLES) Definition 25 clearly only states dominance as going *from* holes to roots.

   (d) (NO EMPTY FRAGMENTS) Since holes can only introduced via $(\multimap_e)$-labels (only (5.4) and (5.7) may introduce an unlabelled dominance variable), and *not* via dominance atoms, there may not be any variable not appearing as the head of a labelling atom (that is, a hole), without having been itself introduced by a non-zero arity labelling atom (as child, then).

□

5.4.2 Polarity

This section utilises the results on polarised Glue Semantics (§3.5.3). We use polarity of formulas' occurrences as introduced in definition 9. with the same convention about polarised "formulas". [16]

In fact, we may now speak of the polarity of a particular *node* (viz. a *variable* from $\mathsf{fragm}(\mathcal{A}_j)$) from a fragment F_j, since all variables from $\mathsf{fragm}(\mathcal{A}_j)$ are injectively mapped to occurrences of subformulas of $\mathcal{A}_j$: we will write $\mathsf{pol}(X) := \mathsf{pol}(\mathsf{ti}(X))$ for $X \in \mathcal{V}(\mathcal{A}_j)$, thus avoiding confusion between *occurrence* and *formula*.

A straightforward corollary of definition 9 and the structural lemma 9 thus assigns polarities to the nodes—via type inference:

Corollary 14 *Let F_j be a fragment and $X \in \mathcal{V}(F_j)$ a node thereof. If X is*

- *a* **hole**, *then* $\mathsf{pol}(\mathsf{ti}(X)) = -$;
- *a* **backbone** *node, then* $\mathsf{pol}(\mathsf{ti}(X)) = -$;
- *a* **hypothesis** *node, then* $\mathsf{pol}(\mathsf{ti}(X)) = +$;
- *labelled with $(\multimap_i)_\varphi$, then $\mathsf{pol}(\varphi) = -$ (this particular occurrence of φ in $\mathcal{A}_j$ has no antecedent under* ti*) and $\mathsf{pol}(X) = +$.*

All possible nodes of a fragment are thereby "polarised" (cf. figure 5.15).

Proof. Easy left-leaf-root induction utilising properties of type inference (proposition 7). □

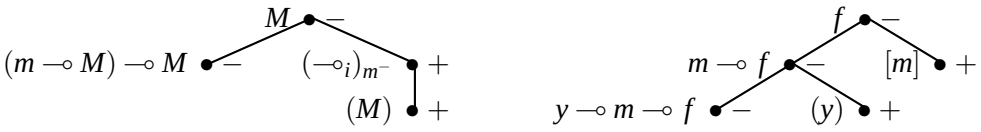


Figure 5.15: Distribution of polarities among graph nodes from fragments (QNP and sentence-embedding verb, cf. figure 5.5).

Remark. A rereading of the structural definitions from §3.5.2 (7 and 6) then also yields as a corollary: For a fragment F_j, $\mathsf{pol}(\varphi) = +$ iff $\varphi \in \mathsf{Prm}_r(\mathcal{A}_j)$ and $\mathsf{pol}(\varphi) = -$ iff $\varphi \in \mathsf{Suf}(\mathcal{A}_j)$

A more useful version of the polarity pattern is given by the following corollaries, that thus leave little combinatoric liberty for dominances between fragments (cf. definition 25):

Corollary 15 *In any satisfiable Glue derivation and if φ is not a variable,*

1. *For any fragment that is a producer for φ, there is a unique consumer for φ;*

[16] Subformulas of an axiom are indeed very likely to be found in several occurrences, not least in the case of **replacers** (definition 23), which oppose polarities of the resource they replace.

2. *For any fragment that is a consumer for φ, there is a producer for φ.*[17]

Corollary 16 *For any fragment that is a replacer for φ, there are both a replacer for φ and a unique consumer for φ.*

Proof.

1. Let F_j be a consumer for ψ. Let then $X = X_j^{uv}$ ($u \in 1^*2$ and $v \in 1^*$) be a hole having per definition an ancestor $Y := X_j^{uv'}$, $v' \leq v$, bearing label $(\multimap_i)_\psi$. Following the proof of lemma 11, we see that there exists a non-ψ-modifier fragment F_i and a substitution σ such that the ancestor Y of X verifies $\sigma(\mathsf{ti}(Y)) = \varphi' \in \mathsf{Suf}(\mathcal{A}_i)$ (a lower ancestor of X (than X_j^u) obviously maps a smaller suffix of $\mathcal{A}_i$ (than $\sigma(\mathsf{ti}(X_j^u))$)).

   Let $\varphi := \mathsf{ti}(Y) := \mathsf{ti}(X_j^{uv'}) = \psi \multimap \chi$ with $\chi = \mathsf{ti}(X_j^{uv'1})$. Because of its $(\multimap_i)$-status, ψ cannot contain any variable (assumption 6), therefore $\varphi' = \psi \multimap \sigma(\chi) \in \mathsf{Suf}(\mathcal{A}_i)$. Besides, the F_j occurrence of ψ is negative (lemma 15 for $(\multimap_i)_\psi$).

   Now, F_i being a non-ψ-modifier yet containing another occurrence of ψ, this other occurrence of ψ must be positive, and unique in F_i (**polarity pattern**).

   And because $\varphi'^- = \psi^+ \multimap \sigma(\chi)^-$ is a suffix of $\mathcal{A}_i$, there is a node $Y' := X_i^{w1}$ s.t. $w \in 1^*$ and $\mathsf{ti}(Y') = \varphi'^-$ (after lemma 9, all suffixes of a given axiom have a backbone antecedent under ti); Its sibling $Z' := X_i^{w2}$ then verifies $\mathsf{ti}(Z') = \psi^+$. [18]

   But then Z' must be labelled with $[\psi^+]$, because its only other occurrence (no need to take subsumption into account since it does not contain any variable) in a non-ψ-modifier is a right-branch premiss $X_j^{uv'1}$, i.e. is contained in $\mathsf{Prm}_r(\mathcal{A}_j)$. This is disjoint with $\mathsf{Suf}(\mathcal{A}_j)$ for $\mathcal{A}_j$ is a non-ψ-modifier (as a consumer for ψ). Therefore only (5.4) may apply: $\langle$QED$\rangle$.

2. Let F_i be a producer for φ. Let then $X = X_i^{u2}$ ($u \in 1^*$) be its $[\varphi^+]$-labelled node, thus the only positive occurrence of φ in F_i.

   There must then be another unique fragment with a unique negative occurrence of φ (**polarity pattern**), let us call it F_j.

   X stems from (5.4) in the definition 19 of fragm, which implies that φ does not subsume any suffix of any non-φ-modifier. This again implies that φ's occurrence in F_j cannot be a backbone variable, but necessarily in a right-branching premiss. Since all suffixes of right-branching premisses are of positive polarity, the missing negative occurrence of φ appears in one of the ψs of $(\multimap_i)_\psi$-labels.

   But we show it must actually *be* one of these ψs: Were it a proper subformula of a ψ (a proper *suffix* for polarity reasons), the first part of this corollary

[17] We conjecture such a producer could be proved unique *modulo subsumption*. However, the final soundness proof just requires its existence.

[18] In fact, one could prove that the word w is exactly v; that is, the right-branch path in $\mathcal{A}_j$ is really one-to-one mapped onto a backbone path in $\mathcal{A}_j$ (cf. figure 5.11). This is where the correspondence between fragments evoked in lemma 11 is so strong.

proved above would compel ψ to appear as a $[\psi]$-labelled node in another unique fragment. But then the fragment would also be F_i, for it would have a positive occurrence of φ: as $[\psi^+ = \chi^- \multimap \varphi^+]$.

So this occurrence could not be as $[\varphi]$, although it *be* unique: this is a contradiction.

Altogether: F_j is a consumer for φ, $\langle \text{QED} \rangle$.

□

Proof. Similar reasoning as in the proofs above yields unicity of the producer and existence of the consumer for any replacer. □

A similar reasoning also constrains those dominance edges that do not arise from hypothesis dependency:

Corollary 17 *For every $(\varphi, \top)$-consumer there is a $(\varphi', \top)$-producer where g' subsumes g.*

Proof. The setting is the same as in the proof above, but the ancestor $Y := X_j^u$ ($u \in 1^*$) and the hole X_j^{uv} are one and the same: $v = \varepsilon$. There is again a non-ψ-modifier $\mathcal{A}_i$ s.t. $\sigma(\mathsf{ti}(Y)) \in \mathsf{Suf}(\mathcal{A}_i)$, but since $\varphi = \mathsf{ti}(Y)$ is an atom or a variable (easily inferred from definition 19), so is $\sigma(\mathsf{ti}(Y))$, which is either φ itself if atomic, or $\sigma(X)$ if $\varphi = X$, thus in any case subsumed by φ.

Therefore, $\sigma(\varphi) = \mathcal{R}(\mathcal{A}_i)$ (under definition 19 only the last suffix of an axiom cannot be split), and $\mathcal{A}_i$ is a $(\varphi', \emptyset)$-producer. □

5.4.3 Soundness

Firstly, we see how partial weak soundness (proposition 18) almost directly follows from weak local soundness (proposition 10, §5.3.3) and the results from §5.4.1.

It roughly means that "everything is correct up to hypothesis book-keeping". As such it does *not* need the polarity pattern from subsection 5.4.2, that constraints the *hypothesis* behavior of fragments.

Proposition 18 (Partial weak soundness) *Let $\mathcal{A}$ be a satisfiable sequence of Glue axioms and $(\mathcal{M}_\tau, \alpha)$ a constructive solution of $\mathsf{T}(\mathcal{A})$. Then τ is a partial weakly correct derivation tree.*

Proof. Let $\mathcal{A}$ be a sequence of Glue axioms and $(\mathcal{M}_\tau, \alpha)$ a constructive solution of $\mathsf{T}(\mathcal{A})$. We prove by structural induction on τ that $\mathsf{F}(\tau)$ is defined:

1. If $\tau = (\text{ax})_j$, then $\mathsf{ft}(\tau)$ is always defined (definition 13), and then so is $\mathsf{F}(\tau)$ (definition 17).
2. If $\tau = [\varphi]$, then likewise, $\mathsf{ft}(\tau)$ is defined.
3. If $\tau = (\multimap_e)(\tau_1, \tau_2)$ with τ_1 and τ_2 partial correct derivations trees (Induction Hypothesis), then $\mathsf{F}(\tau)$ is by definition 17 well-defined if $\begin{cases} \mathsf{F}(\tau_1) = \psi \multimap \chi \\ \text{and } \mathsf{F}(\tau_2) = \psi' \end{cases}$, where ψ' subsumes ψ, (in which case $\mathsf{F}(\tau) = \mathsf{mgu}(\psi, \psi')(\chi)$).

Since we have a **constructive solution**, let us choose X, Y and Z such that $\alpha(X) = \tau$, $\alpha(Y) = \tau_1$ and $\alpha(Z) = \tau_2$. Hence, necessarily, $(\mathcal{M}_\tau, \alpha) \models X : (\multimap_e)(Y, Z)$ and $X = X_j^u$ implies $Y = X_j^{u1}$ and $Z = X_j^{u2}$. [19] Proposition 10 (**weak local soundness** for fragment F_j) then says that $\mathsf{F}(\tau_1) = \mathsf{ti}(Y)$ and $\mathsf{F}(\tau_2) = \sigma(\mathsf{ti}(Z))$ for some substitution σ of variables.

According to definition 19, $Z = X_j^{u2}$ is labelled either as $[\psi]$, as $(\multimap_i)_\chi$ if ψ is an implication $\chi \multimap \omega$, or is a hole (without label).

The properties of type inference (proposition 7:3.) yield in any case $\mathsf{ti}(Y) = \mathsf{ti}(Z) \multimap \mathsf{ti}(X)$, so that $\mathsf{F}(\tau_1)$ always is an implication $\psi \multimap \chi$ with $\psi = \mathsf{ti}(Z)$. Then $\mathsf{F}(\tau_2) = \sigma(\mathsf{ti}(Z)) = \sigma(\psi)$: ψ subsumes $\mathsf{F}(\tau_2)$; $\langle$QED$\rangle$.

4. If $\tau = (\multimap_i)_\varphi(\tau')$ with τ' partial correct derivation tree (IH), then $\mathsf{F}(\tau)$ is after definition 17 always defined (no LL-structural conditions in this subcase).

This means that τ is partial weakly correct. □

Secondly, we need here let the polarity results (corollary 15) play their role, finally: They will further ensure that bookkeeping of hypotheses be correct, too. In fact, partial soundness here could mean that "everything is correct up to the number of hanging hypotheses at the root."

Proposition 19 (Partial soundness) *If $\mathcal{A}$ is a satisfiable sequence of Glue axioms and $(\mathcal{M}_\tau, \alpha)$ a constructive solution of $\mathsf{T}(\mathcal{A})$, then τ is a partial correct derivation tree.*

Proof. In the inductive cases checked for partial weak soundness (proposition 18), it suffices to show that not only $\mathsf{F}(\tau)$, but also $\mathsf{ft}(\tau)$, is defined, i.e., that $\mathsf{pr}_2(\mathsf{ft}(\tau))$ is always well-defined: this is a *test on the multiset of hypotheses*.

3. no checking to perform, since the union $A_\mathcal{L} \uplus B_\mathcal{L}$ is always defined, provided $A_\mathcal{L}$ and $B_\mathcal{L}$ are;

4. to be checked: $\mathsf{pr}_2(\mathsf{ft}(\tau')) = \{\varphi\}_\mathcal{L} \uplus A_\mathcal{L}$ (at least one φ must still pend in hypothesis storage) with τ' partial correct derivation tree (IH).

   We know that $\tau = (\multimap_i)_\varphi(\tau')$ is a **partial weakly correct** derivation tree.

   Since $(\mathcal{M}_\tau, \alpha)$ is a **constructive solution** of $\mathsf{T}(\mathcal{A})$, there are variables X_j^u and X_j^{u1} ($u \in 1^*21^*$—lemma 8) with $X_j^u : (\multimap_i)_\varphi(X_j^{u1}) \in \mathsf{T}(\mathcal{A})$. Either X_j^{u1} is a hole, or it is labelled $(\multimap_i)_{\varphi'}(X_j^{u11})$. In the second case, the inductive definition of w_j ((5.6) and (5.7)) ensures that the last such variable introduced will eventually be a hole.

   Let X be the hole, then, s.t. $X = X_j^{v1}$ (with u prefix of v) and $X_j^{v11} \notin \mathcal{V}(F_j)$. Then X_j^v is labelled $(\multimap_i)_\psi(X)$ for some variable-free (assumption 6) ψ.

   This means that fragment F_j is a consumer for φ). Corollary 15 says then that there is a (unique) fragment F_i that is a consumer for φ.

   According to the definition 25 of dominance atoms, there must be then an atom $X \lhd^* Y$ in $\mathsf{T}(\mathcal{A})$, where $Y = \mathcal{R}(F_i)$. In fact, there is such an atom

[19] One can always assume that all three nodes belong to the same fragment, because any non-hole (as X must be for at least one fragment) branches at least one level deeper in the same fragment (again, by definition 19).

to every (root of) replacer for φ as well (corollary 16), and the **polarity pattern** ensures us that all fragments using φ as a hypothesis resource (consumers, producers and replacers) are thereby *exhaustively* considered.

Likewise, all fragments dominated by *this particular hole* X of F_j are thereby considered: Let τ'' be the finite constructor tree rooted in $\alpha(Y)$. We know that $\tau' \lhd^* \tau''$, i.e. there is a *path* between $\alpha(X)$ and $\alpha(Y)$. It is to prove that there are "enough" labels $[\varphi]$ along this path, at any rate at least one just under X.

F_i provides one such label (from its *unique* occurrence of φ as $[\varphi]$ up to its root Y).

Whatever comes in between X and Y is a replacer for φ or some other fragment *not involving* φ, but may not be a consumer for φ, since this one is unique. Yet a φ-replacer is not bothering either: it will also pass a $[\varphi]$-labelled node up to the root, as well as "consume" the other occurrence of φ in form of its $(\multimap_i)_\varphi$-labelled node (cf. fragment for *sleepy* in figure 5.14).

All this ensures that $\mathsf{ft}(\tau)$ is fully defined, making τ a correct partial derivation tree as well. □

Looking back at the consequence of the polarity pattern (corollary 15), one notices that only the first direction has been used (i.e. the "top-down" version: unicity of consumers to a given producer). The "bottom-up" version, that any hypothesis producer admits a unique consumer, will be, logically enough, used by the next and last result, thus closing the soundness issue.

By now, the only thing that could run awry in a solution derivation tree would be there be *too many* hypotheses pending at the root (i.e. when the proof is meant to be over). Classical logic would not be too fussy about that (it merely means some **weakening of hypotheses** is possible, cf. §3.3.1), but this is a major and grounding issue in Linear Logic derivations, which indeed we do not want our translation to let go of.

Theorem 20 (Soundness of the translation) *If $\mathcal{A}$ is a satisfiable sequence of Glue axioms and $(\mathcal{M}_\tau, \alpha)$ a constructive solution of $\mathsf{T}(\mathcal{A})$, then τ is a correct derivation tree from $\mathcal{A}$.*

Proof. From the results above we know that in these conditions, we already know that τ is a correct *partial* derivation tree. To prove remains then that it is plain correct, having bound all its hypotheses when finally reaching its (only) root: $\mathsf{pr}_2(\mathsf{ft}(\tau)) = \emptyset_{\mathcal{L}}$.

Suppose the multiset of hypotheses is not empty, i.e. there is some $\varphi \in \mathsf{pr}_2(\mathsf{ft}(\tau))$. This hypothesis may only have been introduced by a $[\varphi]$-labelled node; let F be a highest fragment containing such a node: $\tau \lhd^* \tau' := \alpha(\mathcal{R}(F))$.

It could be either a replacer or a producer for φ, but in either case then F is again *dominated* by a consumer for φ or a replacer for φ (existing by the second direction of corollary 15 (resp. by corollary 16)). This domination of this fragment G implies that in the solution tree τ there is a node *above* τ passing a label $[\varphi]$ up to the root, i.e. $\tau \lhd^* \tau'' \lhd^* \tau'$ for $\tau'' := \alpha(\mathcal{R}(G))$.

This is in contradiction with F's being a *highest* fragment introducing a label $[\varphi]$: $\langle \text{QED} \rangle$. □

Therefore, the translation is sound if one restricts oneself to satisfiable LL derivations (needed by the last three *global* results, but which is always the case for Glue axiom sets extracted from actual sentences) and under compliance to the **theoretical assumptions** (in particular assumption 1 from [GL98] which does exclude some more elaborated aspects of Glue).

5.5 Conclusion: completeness?

We have proved the translation as defined in the two steps (fragments and dominance, definitions 19 and 25, respectively) above **sound**. That is, of the constraint $\mathsf{T}(\mathcal{A})$ output by the translation, all constructive solutions are indeed correct derivation trees from $\mathcal{A}$ (definition 15).

Another interesting issue upon forging translations is **completeness**: So far we have not proved that all expected readings of an ambiguous sentence are actually accessible via our translation, only that everything that is accessible is a valid reading. So the next theoretical step to achieve would be to prove: Every correct derivation tree from axioms $\mathcal{A}$ is a solution of $\mathsf{T}(\mathcal{A})$.

Since several derivation trees in Glue Semantics do not necessarily yield different meaning terms, a lot of **proof normalisation in Linear Logic** would be involved, so as to consider only the essential derivations from an axiom sequence $\mathcal{A}$, and thus to render a proper notion of solution between both the input and the output of T.

This could not be proved so far; Nevertheless, we obtained **empirical confirmation** that all expected readings are faithfully output by the translation's output solution space. This has been observed on the whole range of linguistic phenomena the translation strives to account for (as described in §3.6), and which its soundness proof relies on (cf. theoretical assumptions, §5.2.1).

Empirical completeness is a first satisfactory step toward a hopefully forthcoming theoretical confirmation.

Chapter 6

Conclusion

6.1 Wrapping up

We described and proved sound a translation from a standard fragment of Glue Semantics. After arguing in favor of underspecification when coping with semantic ambiguity, and the search for correspondences between its instances in several formalisms (chapter 1), we focused on the output formalism of our own contribution: Dominance Constraints (chapter 2). There we stressed in particular the handiness of the more intuitive dominance graphs, and the striking advantage there to exist tractable solving algorithms for these. The second formalism we focused on is the input of the contribution: Glue Semantics (chapter 3). We described the resource-conscious approach it owes to Linear Logic, and how ambiguity is implicitly raised by performing a proof. The link between its proofs and meaning construction is just another instance of the Curry-Howard Correspondence. In chapter 4, we reviewed some similar translations found in the literature, in particular two recent ones producing Dominance Constraints, from well-known and broadly used formalisms: Hole Semantics and Minimal Recursion Semantics. The interest in new translations was all the more evident then, since efficient solving algorithms consequently apply to both. Eventually, in chapter 5 our contribution was exposed. After introducing necessary preliminaries and in particular assumptions on the Glue inputs, we presented the translation itself and some properties thereof. Finally a step-wise soundness proof was offered, along with a few well-formedness properties.

6.2 Open issues, future work

Open issues

The contribution developed in chapter 5 leaves a few issues open, as well as perspectives of further development.

Validity of the polarity pattern. The uniform theory of adverbs developed in [Dal01, chap. 10] provides a counterexample to assumption 1 (cf. footnote 12, §5.4.2): an axiom with *three* occurrences of the same resource. It thus would not be possible to classify the axiom according to definition 23. Yet it was possible,

experimentally, to produce a dominance graph for the sentence it is extracted from. The solutions were the readings expected for the sentence.

A natural interrogation then is whether this problem can be raised, or perhaps the assumption modified (weakened), in order to accommodate this more uniform theory of modification.

Likewise, non-standard extensions of Glue go beyond the implicative fragment of Linear Logic (for the treatment of anaphora, say, or dynamic semantics). It is not clear how assumption 1 could be, if at all, sustained when more LL operators are present.

Completeness. The empirical completeness was assessed, but a proof misses.

We believe the proof is accessible, perhaps via a thorougher proof-theoretical study of our version of Glue. To this extent, an approach that base on the **sequent style** of Glue Semantics could be envisaged. The natural deduction fashion of Linear Logic, exposed throughout chapters 3 and 5, probably gives a clearer global visions of the proofs, but sequent deductions are actually more general: they constitute a kind of metatheory of the resources utilised in a proof ([CvG00]).

In any case, a proper completeness proof involves a thorougher proof-theoretical study, in particular **proof-normalisation** ([CvG00]).

Simplification of the translation We conjecture another, simpler version a different translation could be stated, that would greatly simplify the proving of soundness. This translation would produce only compact fragments: by outputting for each axiom the conclusion as the root and each premiss as a direct child. This way, the translation would be reduced to the strict transcription of the scopal behavior, without interfering from the meaning part.

Arguably, though, we foresee it could displace the complexity of the soundness proof towards the completeness issue.

Axiomatic weakening of theoretical assumptions. It would be interesting to study more in details the proofs from chapter 5 in order to reduce them to their most general version: In the axiomatic tradition, try to determine which part of each assumption is perhaps provable, and which is not. For instance, could the central assumption 1 be actually *inferred* from theorem 1?

Hypernormal connexity. We already saw the relevance of the Net Hypothesis for Dominance Constraints (chapter 4). To this extent, it would be interesting to prove the output graphs $\mathsf{T}(\mathcal{A})$ hypernormally connected, as a further corroborating step of the Net Hypothesis.

All implemented outputs of our translation (cf. Utool in §2.3.3) were nets, and we conjecture a proof may confirm this empirical observation.

Bibliography

[AC02a] A. Asudeh and R. Crouch. Coordination and Parallelism in Glue Semantics: Integrating Discourse Cohesion and the Element Constraint. In Miriam Butt and Tracy Holloway King, editors, *Proceedings of the LFG02 Conference*, CSLI publications, National Technical University of Athens, Athens, 2002.

[AC02b] A. Asudeh and R. Crouch. Glue semantics for HPSG. In F. van Eynde, L. Hellan, and D. Beermann, editors, *on-line Proceedings of the HPSG '01 Conference*, Norwegian University of Science and Technology, Trondheim, Norway, 2002.

[Als92] H. Alshawi, editor. *The Core Language Engine*. MIT Press, Cambridge/London, 1992.

[BDNM04] Manuel Bodirsky, Denys Duchier, Joachim Niehren, and Sebastian Miele. A new algorithm for normal dominance constraints. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.

[Bos96] Johan Bos. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, 1996.

[Bos02] Johan Bos. *Underspecification and resolution in discourse semantics*. PhD thesis, Saarland University, 2002.

[CF00] Ann Copestake and Dan Flickinger. An open-source grammar development environment and broad-coverage english grammar using HPSG. In *Conference on Language Resources and Evaluation*, 2000.

[CFS01] A. Copestake, D. Flickinger, and I. Sag. Minimal Recursion Semantics. An Introduction. *Language and Computation*, 1(3):1–47, 2001.

[CFvG99] R. Crouch, A. Frank, and J. van Genabith. Glue, underspecification and translation. In H. Bunt and R. Muskens, editors, *Computing Meaning*, volume 2. Kluwer Academic Press, 1999.

[CLF01] A. Copestake, A. Lascarides, and D. Flickinger. An algebra for semantic construction in constraint-based grammars. In *Proceedings of the ACL-01*, Toulouse, France, 2001.

[CvG99] R. Crouch and J. van Genabith. Context change, underspecification, and the structure of glue language derivations. In Dalrymple [Dal99], pages 117–189.

[CvG00] R. Crouch and J. van Genabith. Linear logic for linguists. Introductory course to *ESSLLI-00*, 2000.

[Dal99] M. Dalrymple, editor. *Semantics and Syntax in Lexical Functional Grammar: The Resource Logic Approach*. MIT Press, Cambridge, MA, 1999.

[Dal01] M. Dalrymple. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics*. Academic Press, 2001.

[DGLS99] M. Dalrymple, V. Gupta, J. Lamping, and V. Saraswat. Relating resource-based semantics to categorial semantics. In Dalrymple [Dal99], pages 261–280.

[DKIZ95] M. Dalrymple, R. Kaplan, J.T. Maxwell III, and A. Zaenen, editors. *Formal Issues in Lexical-Functional Grammar*. Stanford Unifersity, 1995.

[DKMZ95] M. Dalrymple, R. Kaplan, J.T. Maxwell, and A. Zaenen, editors. *The Formal Architecture of Lexical-Functional Grammar*, chapter 1, pages 7–28. In Dalrymple et al. [DKIZ95], 1995.

[DLS93] M. Dalrymple, J. Lamping, and V. Saraswat. LFG Semantics via Constraints. In *Proceedings of the Sixth Conference of the European Association for Computational Linguistics*, 1993.

[EKN01] M. Egg, A. Koller, and J. Niehren. The constraint language for lambda structures. *Journal of Logic, Language, and Information*, 10:457–485, 2001.

[FKNT04] Ruth Fuchss, Alexander Koller, Joachim Niehren, and Stefan Thater. Minimal recursion semantics as dominance constraints: Translation, evaluation, and analysis. In *Proceedings of the 42nd ACL*, Barcelona, 2004.

[FKT05] Dan Flickinger, Alexander Koller, and Stefan Thater. A new well-formedness criterion for semantics debugging. In *Proceedings of the 12th International Conference on HPSG*, Lisbon, 2005.

[Gal91] Jean Gallier. Constructive Logics. Part II: Linear Logic and Proof Nets. Technical report, CIS Department, University of Pennsylvania, 1991. URL: `ftp://ftp.cis.upenn.edu/pub/papers/gallier/conslog2.ps`.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 45:1–102, 1987.

[Gir95] J.-Y. Girard. Linear logic: its syntax and semantics. In J.-Y. Girard, Y. Lafont, and L. Régnier, editors, *Advances in Linear Logic*. Cambridge University Press, 1995.

[GL98] Vineet Gupta and John Lamping. Efficient linear logic meaning assembly. In *Proceedings ACL'98*, pages 464–470, Montreal, 1998.

[Gri90] Timothy G. Griffin. A Formulae-as-Types Notion of Control. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, San Fransisco, CA, 1990. ACM Press.

[Kel88] William R. Keller. Nested Cooper Storage: The proper treatment of quantification in ordinary noun phrases. In U. Reyle and C. Rohrer, editors, *Natural Language and Linguistic Theories*, volume 35 of *Studies in Linguistics and Philosophy*, pages 432–447. Reidel, 1988.

[KMN00] Alexander Koller, Kurt Mehlhorn, and Joachim Niehren. A polynomial-time fragment of dominance constraints. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics*, pages 368–375, 2000.

[KNS00] A. Koller, J. Niehren, and K. Striegnitz. Relaxing underspecified semantic representations for reinterpretation. *Grammars*, 3(2-3), 2000.

[KNT03] Alexander Koller, Joachim Niehren, and Stefan Thater. Bridging the gap between underspecification formalisms: Hole semantics as dominance constraints. In *Proceedings of the 11th EACL*, Budapest, 2003.

[Kol04] Alexander Koller. *Constraint-based and graph-based resolution of ambiguities in natural language*. PhD thesis, Universität des Saarlandes, 2004.

[KR93] Hans Kamp and Uwe Reyle. *From Discourse to Logic*. Kluwer, Dordrecht, 1993.

[Kri03] J.-L. Krivine. Dependent choice, 'quote' and the clock. *Theoretical Computer Science*, 308:259–273, 2003.

[Lev05] Iddo Lev. Decoupling Scope Resolution from Semantic Composition. In *Proc. of the 6th International Workshop on Computational Semantics (IWCS-6)*, pages 139–150, 2005.

[Mon74] Richard Montague. The proper treatment of quantification in ordinary English. In R. Thomason, editor, *Formal Philosophy. Selected Papers of Richard Montague*. Yale University Press, New Haven, 1974.

[Per00] Guy Perrier. From intuitionistic proof nets to interaction grammars. In *Proceedings of the 5th TAG+ Workshop*, Paris, 2000.

[Rey93] Uwe Reyle. Dealing with ambiguities by underspecification: construction, representation, and deduction. *Journal of Semantics*, 10:123–179, 1993.

[vGC99] J. van Genabith and R. Crouch. Dynamic and underspecified semantics for LFG. In Dalrymple [Dal99], pages 209–260.