From perception to communication
An analysis of meaning and action using a theory of types with records (TTR)

Robin Cooper

Draft, December 8, 2020
PLEASE QUOTE WITH CARE

Todo list

Contents

Acknowledgements

I am grateful to many people for discussion which has led to significant changes in this material. Among them are: Ellen Breitholtz, Liz Coppock, Simon Dobnik, Elisabet Engdahl, Tim Fernando, Jonathan Ginzburg, Eleni Gregoromichelaki, Torbjörn Lager, Staffan Larsson, Andy Lücking, Bill Noble, Bengt Nordström, Aarne Ranta, Hannes Rieser, Asad Sayeed, Kwong-Cheong Wong. None of these people is responsible for what I have done with their ideas and suggestions.

This research was supported in part by the following projects: Records, types and computational dialogue semantics, Vetenskapsrådet, 2002-4879, Library-based grammar engineering, Vetenskapsrådet, 2005-4211, Semantic analysis of interaction and coordination in dialogue (SAICD), Vetenskapsrådet, 2009-1569, and Vetenskapsrådet project 2014-39 for the establishment of the Centre for Linguistic Theory and Studies in Probability (CLASP) at the University of Gothenburg.

Introduction

Comments in introttl.pdf

Where to discuss neural TTR?

As we interact with the world and with each other we need to classify objects and events, that is, we need to make judgements about what types of objects and events we are confronted with. This is an important part of what is involved in planning the future actions we should carry out and how we should coordinate with other agents in carrying out collaborative actions. This is true of action in general, including linguistic action. The aim of this book is to characterize a notion of type which will cover both linguistic and non-linguistic action and to lay the foundations for a theory of action based on these types. We will argue that a theory of language based on action allows us to take a perspective on linguistic content which is centered on interaction in dialogue and that this is importantly different to the traditional view of natural languages as being essentially similar to formal languages such as logics developed by philosophers or mathematicians. At the same time we will argue that the tremendous technical advances made by the formal language view of semantics can be incorporated into the action-based view and that this can lead to important improvements both of intuitive understanding and empirical coverage.

Part I of the book (Chapters 1–3) deals with a theory of types related to perception and action and shows a way of presenting a theory of grammar within a theory of action. Part II (Chapters 4–7) then looks at a number of central issues in semantics from a dialogical perspective and argues that there are advantages to looking at some old puzzles from this perspective.

In Chapter 1 we introduce a notion of perception of an object or event as making a judgement that the object is of a type. In symbols, we write $a : T$ to indicate that object a is of type T. We shall talk interchangeably of an object being of a type or being a witness for a type. Our claim is that we can only perceive something as being of a type, even if that type is very general (like *PhysicalObject* or *Event*) – we cannot perceive it *simpliciter*. We present basic notions of the theory of types which will be developed in the book, TTR, a type theory with records, which builds to a great extent on ideas taken from the type theory of Per Martin-Löf although we have made significant changes both in the general design and aims of the theory and a number of details which appear to us to be motivated by cognitive and linguistic considerations. The overall approach presented here owes much to the theory of situations and situation semantics presented by Barwise and Perry in the 1980's. One of the themes of this book is a working out of parts of the old situation theory using ideas taken from Martin-Löf's type theory.

A central notion in TTR is that of *record*. The term "record" is used in computer science for what is often called an attribute-value matrix (AVM) or feature structure in linguistics. A record is a collection of fields consisting of a label (attribute or feature in the standard linguistic way of talking) and an object of some kind (which itself can be a record). An schematic example of a record is given in (1), where the ℓ_i are labels and the o_i are objects.

(1) $$\left[\begin{array}{lll} \ell_0 & = & \left[\begin{array}{lll} \ell_1 & = & o_0 \\ \ell_2 & = & o_1 \end{array}\right] \\ \ell_3 & = & o_2 \end{array}\right]$$

Records are witnesses for record types which are also collections of fields. Rather than objects, the fields in a record type contain types. In the schematic example in (2) the T_i are types.

(2) $$\left[\begin{array}{lll} \ell_0 & : & \left[\begin{array}{lll} \ell_1 & : & T_0 \\ \ell_2 & : & T_1 \end{array}\right] \\ \ell_3 & : & T_2 \end{array}\right]$$

The record (1) will be of the type (2) just in case the objects are of the types with the same labelling, that is, $o_0 : T_0$, $o_1 : T_1$ and $o_2 : T_2$. Martin-Löf's orginal type theory did not have records or record types though there have been many suggestions in the literature on how to add them. We have borrowed freely from some of these ideas in TTR although the way we have developed the notions differs essentially from previous proposals. We will use records and record types to model situations and situation types.

In Chapter 2 we introduce some basic notions of a theory of action based on these types which will be developed further as the book progresses and apply the theory of types from Chapter 1 to basic notions of information update in dialogue. Here we build on seminal work on dialogue analysis by Jonathan Ginzburg and also related computational implementation by Staffan Larsson leading to the information state update approach to dialogue systems. We have adapted these ideas in a way that allows us to pursue the questions of grammar and semantics that we take up in the remainder of the book. A central notion here is that of the dialogue gameboard which we construe as a type of information state representing the current state of play in the dialogue from the perspective of a dialogue participant. It includes the dialogue participant's view of what has been committed to as being true in the dialogue so far and what questions are currently under discussion.

In Chapter 3 we show how syntax and semantics can be embedded in the theory of action characterized in Chapters 1 and 2. This is in contrast to a formal language view where language is seen as a set of analyzed strings of symbols associated with meanings of some kind. The philosophical ground of the action-based approach goes back to the relational theory of meaning introduced in Barwise and Perry's situation semantics which focusses on the relation between utterance situations and described situations. This was perhaps the first attempt to generalize the Speech Act Theory developed by Austin and Searle to the concerns of compositional interpretation of syntactic structure. A more recent theory to which the ideas in this chapter are related is that of Dynamic Syntax (DS). While the particular formulations in our approach look rather different from those in DS the two theories have common aims relating to the analysis of language as action and an emphasis on the incremental nature of language which in this chapter we relate to the building of a chart type. There is also a common interest in the treatment of language as a system in flux where an act of speaking can create a new previously unavailable linguistic resource that can be reused in future speech events.

The theory of types that we employ gives us two important notions which will be important in the development of semantics in Part II. The first is the notion of *intensionality*. Types in TTR are intensional in that the identity of a type is not established in terms of the set of objects which are of that type. That is, types are not *extensional* in the way that sets are in a standard set theory. The axiom of extensionality in standard set theory requires that there cannot be two sets which have the same members. In contrast, there can be different types which have exactly the same set of witnesses. The second notion has to do with the facts that the types themselves are treated as objects that can enter into relations and be used to construct new types. We will call this *first class citizenship of types*, though it is related to notions of *intentionality* (with a "t") and *reflection* in programming languages, that is, the ability not only to carry out procedures but to reflect on and reason about them. In our terms, an important enabling factor for human language is that we not only can perceive objects and events in terms of types and act on these perceptions but that we can also reason about and act on the types themselves, for example, in ascribing them to other agents as beliefs or making a plan to achieve a goal by creating an event of a certain type. The types become cognitive *resources* which we can exploit in our communicative activity. In Part II we will look at a number of examples of this.

In Chapter 4 we examine reference by uses of proper names and occurrences of pronouns which are not bound by quantifiers. In order to account for this we need a notion of *parametric content*, which is to say that the content of an utterance depends on a context belonging to a certain type. For example, an utterance of the proper name *Sam* requires a context in which there is an individual named "Sam". But where in her resources should a dialogue participant look for such a context? One obvious place is the conversational gameboard that we introduced in Chapter 2. That is, the dialogue participant should determine whether there has been reference to somebody of that name already in the current dialogue according to her gameboard. Another place is the visual scene, or more generally the ambient situation which the agent can perceive by different sense modalities. This we also represent as a resource using a type – that is, the type for which the ambient situation would be a witness if the agent's perception is correct. Yet another place to look is the agent's long term memory (which we will equate with the agent's beliefs, although one may ultimately wish to make a distinction). This resource is also modelled as a type representing how the world would be if the agent's memory or beliefs are correct. The fact that we are reasoning about to what extent the context type associated with the utterance matches the types modelling the agent's relevant resources enables us to talk about cases where there are names of non-existent objects (that is, the agent's resource types do not exactly match the world) or where a single object in the world corresponds to two objects in the resources or *vice versa* (another way in which there can be a mismatch between reality and an agent's resources).

In Chapter 5 we look at frames associated with common nouns. The idea of frames goes back to early work on frame semantics by Fillmore and also psychological work on frames by Barsalou. We will construe frames as situations (modelled as records in TTR). We will argue that frame types are an additional kind of resource which is exploited in natural language semantics. A common noun like *dog*, in addition to being associated with the property of being a dog, can also be associated with a type of situation (a frame type) which is common for dogs, for example, where the dog has a name, an age and various other attributes we commonly attribute to dogs. We will argue that such a frame can play an important role in interpreting utterances such as *the dog is nine* in the sense of "the dog is nine years old". Some nouns, such as *temperature*, seem to represent frame level predicates, following an analysis suggested by Sebastian Löbner in order to account for the analysis of utterances like *the temperature is rising* where it is not the case that some particular temperature is rising (say, 30°) but that different situations (frames) with different temperatures are being compared. Nouns which normally predicate of individuals can be coerced to predicate of frames. An example is the noun *ship* in an example originally discussed by Manfred Krifka: *four thousand ships passed through the lock* which can either mean that four thousand distinct ships passed through the lock or that there were four thousand ship-passing-through-the-lock events some of which may have involved the same ship. We argue that in order to interpret such examples you need to have as a resource an appropriate frame type associated with the noun *ship*.

In Chapter 6 we explore phenomena in natural language which are standardly referred to as *modality* and *intensionality*. We argue that types as we conceive them are better placed to deal with these phenomena than possible worlds that are used in standard formal semantics. In standard formal semantics propositions are regarded as sets of possible worlds. For example, the proposition corresponding to *a boy hugged a dog* is the set of all logically possible worlds in which a boy hugged a dog is true. What we substitute for this is the type of situations in which a boy hugged a dog. At an intuitive level these notions are quite similar. They both represent mathematical objects which allow for many different possibilities as long as the fact that a boy hugged a dog is held constant across them. One important difference is that sets of possible worlds are extensional sets whereas as our types are intensional. Thus it is possible for us to have two distinct types which have exactly the same witnesses. One pair of such examples we discuss is *Kim sold Syntactic Structures to Sam* and *Sam bought Syntactic Structures from Kim*. Intuitively we want these to represent different propositions and we argue that they can yield different truth conditions when embedded under a predicate like *legal*. (Under Swedish law, for example, it is illegal to buy sex but legal to sell sex.) Another pair involves so-called mathematical propositions which are true in all possible worlds but which nevertheless we would want to represent different propositions: *Two plus two equals four* and *Fermat's last theorem is true* (as proved by Andrew Wiles).

The chapter begins with a discussion of the problems associated with possible worlds analyses. We then continue with a discussion of modality and in particular of how Angelika Kratzer's notions of conversational background and ideals can be seen with advantage as resources based on types and the kind of topoi that Ellen Breitholtz has introduced in the TTR literature. In the third part of the chapter we discuss what are traditionally regarded as intensional constructions involving attitude verbs like *believe* and intensional verbs like *need* and *want*. We treat 'believe' as a relation between individuals and types (corresponding to the content of the embedded sentence). For an individual to believe a type it has to be the case that the type matches (in a way we make precise) the type which models the beliefs (or long term memory) of the individual, that is the same resource that was needed in Chapter 4 to get the dialogical analysis of proper names to work out.

In Chapter 7 we look at generalized quantifiers from the perspective of dialogic interaction. Traditionally generalized quantifiers are treated as sets of sets or sets of properties and the work of Barwise and Cooper on generalized quantifiers built on this idea. Barwise and Cooper also introduced the auxiliary notion of witness set for quantifiers under the heading "Processing quantified statements". In this chapter we turn things around and make the characterization of witness sets the primary notion in defining quantifiers. This makes it more straightforward to account for the anaphoric possibilities relating to quantified expressions in dialogue. We often use quantified statements in dialogue when we have inadequate information to determine their truth. This is particularly true of determiners like *every* and *most* when talking about large sets. We suggest that this phenomenon can be analyzed by estimating a probability based on the evidence presented in our cognitive resources (long-term memory or beliefs as discussed in Chapters 4 and 6). Finally, we give an account of how TTR types can be used to talk of content which is underspecified for quantifier scope. The idea is to compute content types which have the various available contents which can be associated with an utterance as witnesses.

There are a number of general themes which are woven together in this book and which have different emphasis at different points in the text:

- language as action
- linguistic content grounded in perception as type judgement
- language as interaction and coordination
- language as a system in flux
- types, not possible worlds
- types and reference to non-existent objects
- types and cognitive resources
- getting the balance right between language, the external world and mental states

- types as a way of doing underspecification
- avoiding an intermediate "semantic" language such as logical form or discourse representation language but rather giving a direct interpretation of linguistic events in terms of a semantic universe containing structured objects

Behind all this is a desire to find a theory of types which can be used to talk about cognition in general as well as allow us to give a general account of language which includes many of the insights we have gained from separate linguistic theories, a foundation for a formal approach to cognition, if you like.

Why try to do all of this at once? Would it not have been better to write individual books and papers on each of these topics in turn? These are questions that I have asked myself at various points while writing this book. It worries me (and it will probably worry you) despite the fact that I know the answer: it is important to have a single approach to language in which all these issues can be addressed simultaneously. Taking the issues one at a time is not as convincing or ultimately as interesting as showing how these different aspects of language interact in a complex system, giving us a view of linguistic interpretation which both embraces an action oriented approach and preserves the insights we have gained from formal semantics as well as addressing some of the puzzles that it failed to solve adequately.

Part I

From perception and action to grammar

Chapter 1

From perception to intensionality

1.1 Introduction

When we perceive objects and events we classify them as belonging to some type. In this chapter we will explain this idea and introduce a mathematical theory of some of the types we use. Perception and the use of natural language for communication are closely linked. Types, particularly types of events, are a good model for what are often called "propositions", true if there is something of the type and false if there is nothing of the type. It seems that the origin of linguistic meaning seems closely related to perception. If you are talking to a two year old child, you tend to point at physical objects or observable events and utter the corresponding word or phrase. You do not talk about abstract concepts like 'university', 'democracy', 'thought' or 'feeling'. These come later. We will argue in this book, that humans have built on the basic perceptual apparatus in terms of types in a way that allows them to reason about the types themselves and that it is this ability which allows us to talk about intensional notions like 'belief' and 'knowledge'.

In this chapter we will talk first about the notion of perception as type assignment (Section 1.2). We will then lay the basis for our mathematical modelling of types (Section 1.3). We introduce a notion of situation type (Section 1.4) which will be important for the idea that types are used to model propositions (discussed in Section 1.5). Included in the kinds of types we use as situation types are types constructed from predicates and their arguments (*ptypes*) and types which are collections of labelled fields (*record types*).

1.2 Perception as type assignment

Kim is out for a walk in the park and sees a tree. She knows that it is a tree immediately and does not really have to think anything particularly linguistic, such as "Aha, that's a tree". As a human being with normal visual perception, Kim is pretty good at recognizing something as a tree when she sees it, provided that it is a fairly standard exemplar, and the conditions are right: for example, there is enough light and she is not too far away or too close. We shall say that

Kim's perception of a certain object, a, as a tree involves the ascription of a type *Tree* to a. In terms of the kind of type theory discussed by Martin-Löf (1984); Nordström *et al.* (1990), we might say that Kim has made the *judgement* that a is of type *Tree* (in symbols, a : *Tree*).

Objects can be of several types. An object a can be of type *Tree* but also of type *Oak* (a subtype of *Tree*, since all objects of type *Oak* are also of type *Tree*) and *Physical Object* (a supertype of *Tree*, since all objects of type *Tree* are of type *Physical Object*). It might also be of an intuitively more complicated type like *Objects Perceived by Kim* which is neither a subtype nor a supertype of *Tree* since not all objects perceived by Kim are trees and not all trees are perceived by Kim.

There is no perception without some kind of judgement with respect to types of the perceived object. When we say that we do not know what an object is, this normally means that we do not have a type for the object which is narrow enough for the purposes at hand. I trip over something in the dark, exclaiming "What's that?", but my painful physical interaction with it through my big toe tells me at least that it is a physical object, sufficiently hard and heavy to offer resistance to my toe. The act of perceiving an object is perceiving it *as* something. You cannot perceive something without ascribing some type to it, even if it is a very general type such as *thing* or *entity*.

Recognizing something as a tree may be immediate and not involve conscious reasoning. Recognizing a tree as an aspen, an elm or a tree with Dutch elm disease may involve closer inspection and some conscious reasoning about the shape of the leaves or the state of the bark. For humans the relating of objects to certain types can be the result of a long chain of reasoning involving a great deal of conscious effort. But whether the perception is immediate and automatic or the result of a conscious reasoning process, from a logical point of view it still seems to involve the ascription of a type to an object.

The kind of types we are talking about here correspond to pretty much any useful way of classifying things and they correspond to what might be called properties in other theories. For example, in the classical approach to formal semantics developed by Montague (1974) and explicated by Dowty *et al.* (1981) among many others, properties are regarded not as types but as functions from possible worlds and times to (the characteristic functions of) sets of entities, that is, the property *tree* would be a function from possible worlds and times to the set of all entities which are trees at that world and time. Montague has types based on a version of Russell's (1903) simple theory of types but they were "abstract" types like *Entity* and *Truth Value* and types of functions based on these types rather than "contentful" types like *Tree*. Type theory for Montague was a way of providing basic mathematical structure to the semantic system in a way that would allow the generation of interpretations of infinitely many natural language expressions in an orderly fashion that would not get into problems with logical paradoxes. The development of the theory of types which we will undertake here can be regarded as an enrichment of an "abstract" type theory like Montague's with "contentful" types. We want to do this in a way that allows the types to account for content and relate to cognitive processing such as perception. We want our types to have psychological relevance and to correspond to what Gibson (1979) might call *invariants*, that is, aspects that we can perceive to be the same when confronted with similar objects or the same object from a different perspective. In this respect our types are similar to notions developed in situation theory and situation semantics (Barwise and Perry, 1983; Barwise, 1989).

Gibson's notion of attunement is adopted by Barwise and Perry. The idea is that certain organisms are attuned to certain invariants while others are not. Suppose that Kim perceives a cherry tree with flowers and that a bee alights on one of the flowers. One assumes that the bee's experience of the tree is very different from Kim's. It seems unlikely that the bee perceives the tree as a tree in the sense that Kim does and it is not at all obvious that the bee perceives the tree in its totality as an object. Different species are attuned to different types and even within a species different individuals may vary in the types to which they are attuned. This means that our perception is limited by our cognitive apparatus – not a very surprising fact, of course, but philosophically very important. If perception involves the assignment of types to objects and we are only able to perceive in terms of those types to which we are attuned, then as Kant (1781) pointed out we are not actually able to be aware of *das Ding an sich* ("the thing itself"), that is, we are not able to be aware of an object independently of the categories (or types) which are available to us through our cognitive apparatus.

1.3 Modelling type systems in terms of mathematical objects

In order to make our theory precise we are going to create mathematical models of the systems we propose. This represents one of the two main strategies that have been employed in logic to create rigorous theories. The other approach is to create a formal language to describe the objects in the theory and define rigorous rules of inference which explicate the properties of the objects and the relations that hold between them. At a certain level of abstraction the two approaches are doing the same thing – in order to characterize a theory you need to say what objects are involved in the theory, which important properties they have and what relations they enter into. However, the two approaches tend to get associated with two different logical traditions: the *model theoretic* and *proof theoretic* traditions.

The philosophical foundation of type theory (as presented, for example, by Martin-Löf, 1984) is normally seen as related to intuitionism and constructive mathematics. It is, at bottom, a proof-theoretic discipline rather than a model-theoretic one (despite the fact that model theories have been provided for some type theories). However, it seems that many of the ideas in type theory that are important for the analysis of natural language can be adopted into the classical set theoretic framework familiar to linguists from the classical model-theoretic canon of formal semantics starting from Montague (1974). We assume a standard underlying set theory such as ZF (Zermelo-Fraenkel) with urelements (as formulated for example in Suppes, 1960). This is what we take to be the common or garden working set theory which is familiar from the core literature on formal semantics deriving from Montague's original work.

A theory is not very interesting if it does not make *predictions*, that is, by making certain assumptions you can infer some conclusions. This gives you one way to test your theory: see what you can conclude from premises that you know or believe to be true and then test whether the conclusion is actually true. If you can show that your theory allows you to predict some conclusion and its negation, then your theory is *inconsistent*, which means that it is not useful as a scientific theory. One way to discover whether a theory is consistent or not is to formulate it very carefully and explicitly so that you can show mathematical properties of the system and any inconsistencies will appear.

From the informal discussion of type theory that we have seen so far it is clear that it should involve two kinds of entity: the types and the objects which are of those types. (Here we use the word "entity" not in the sense that Montague did, that is, basic individuals, but as an informal notion which includes both objects and types.) This means that we should characterize a type theory with two domains: one domain for the objects of the types and another domain for the types to which these objects belong. Thus we see types as theoretical entities in their own right, not, for example, as collections of the objects which are object of the types. Diagrammatically we can represent this as in Figure 1.1 where object a is of type T_1.

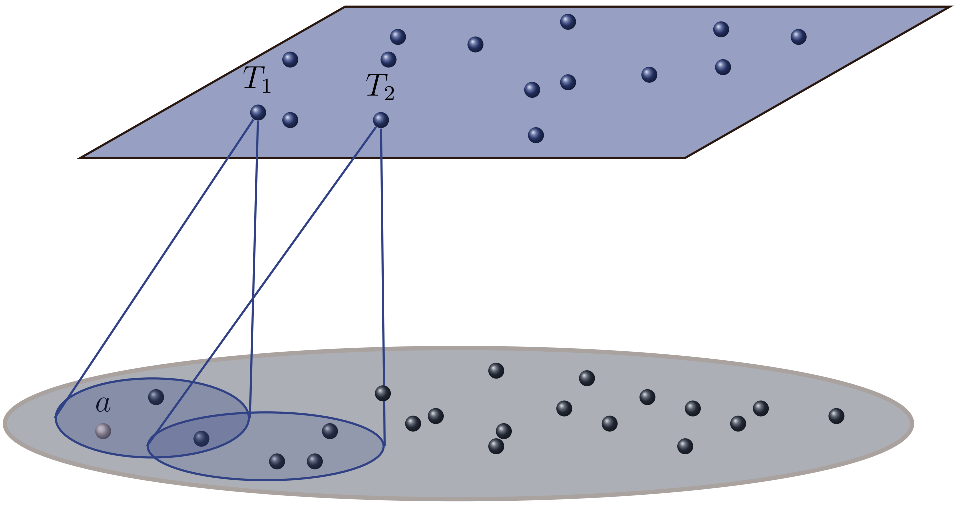


Figure 1.1: System of basic types

A system of basic types consists of a set of types which are *basic* in the sense that they are not analyzed as *complex* entities composed of other entities in the theory. Each of these types is associated with a set of objects, that is, the objects which are of the type, the *witnesses* for the type. Thus if T is a type and $A(T)$ is the set of witnesses for T, then a is of type T (in symbols, $a : T$) just in case $a \in A(T)$. We require that any object a which is a witness for a basic type is not itself one of the types in the system. A type may be *empty* in the sense that it is associated with the empty set, that is, there is nothing of that type.

Notice that we are starting with the types and associating sets of objects with them. This means that while there can be types for which there are no witnesses, there cannot be objects which do not belong to a type. This relates back to our claim in Section 1.2 that we cannot perceive an object without assigning a type to it.

Notice also that the sets of objects associated with types may have members in common. Thus it is possible for objects to belong to more than one type. This is important if we want to have basic types *Elm*, *Tree* and *Physical Object* and say that a single object a belongs to all three types as discussed in Section 1.2.

An extremely important property of this kind of type system is that there is nothing which prevents two types from being associated with exactly the same set of objects. In standard set theory the notion of set is *extensional*, that is sets are defined by their membership. You cannot have two distinct sets with the same members. The choice of defining types as entities in their own right rather than as the sets of their witnesses, means that they can be *intensional*, that is, you can have more than one type with the same set of witnesses. This can be important for the analysis of natural language words like *groundhog* and *woodchuck* which (as I have learned from the literature on natural language semantics) are the same animal. In this case one may wish to say that you have two different words which correspond to the same type, rather than two types with the same *extension* (that is, set of witnesses). Such an analysis is less appealing in the case of *unicorn* and *centaur*, both mythical animals corresponding to types which have an empty extension. If types were extensional, there would only be one empty type (just as there is only one empty set in set theory). In the kind of possible world semantics espoused by Montague the distinction between *unicorn* and *centaur* was made by considering their extension not only in the actual world (where both are empty) but also in all possible worlds, since there will be some worlds in which the extensions are not the same. However, this kind of possible worlds analysis of intensionality fails when you have types whose extensions cannot possibly be different. Consider *round square* and *positive number equal to* $2 - 5$. The possible worlds analysis cannot distinguish between these since their extensions are both empty no matter which possible world you look at.

Finally, notice that there may be different systems of basic types, possibly with different types and different objects. One way of exploiting this would be to associate different systems with different organisms as discussed in Section 1.2. (Later in this book we will see different uses of this for the analysis of types which model the cognitive system of a single agent.) Thus properly we should say that an object a is of type T with respect to a basic systems of types $\mathbf{TYPE}_B$, in symbols, $a :_{\mathbf{TYPE_B}} T$. However, we will continue to write $a : T$ in our informal discussion when there is no danger of confusion.

The definition of a system of basic types is made precise in (1), which is repeated in Appendix 2.

(1) A *system of basic types* is a pair:

TYPE$_B$ = ⟨**Type**, A⟩

where:

1. **Type** is a non-empty set
2. A is a function whose domain is **Type**
3. for any $T \in$ **Type**, $A(T)$ is a set disjoint from **Type**
4. for any $T \in$ **Type**, $a :_{\mathbf{TYPE_B}} T$ iff $a \in A(T)$

Some readers may prefer a slightly less formal characterization which uses the kind of format normally employed in proof theory. This may provide a more easily readable overview of the definitions while suppressing some of the details which are not necessary for intuitive understanding. We will use Γ, normally used for contexts in proof theory, that is sequences of judgements to refer to type systems like those characterized in (1). Thus in (2) Γ corresponds to **TYPE**$_B$. We will write $\Gamma \vdash T \in$ **Type** "$T \in$ **Type** follows from Γ" to represent that **Type** is the set of types in Γ as specified in (1) and that the type T is a member of this set. We will similarly write $\Gamma \vdash a \in A(T)$ to indicate that object a is in the set assigned to T by the function A given by Γ. We can then write a rule as in (2).

(2) For Γ a system of basic types:

$$\frac{\Gamma \vdash T \in \textbf{Type} \qquad \Gamma \vdash a \in A(T)}{\Gamma \vdash a : T}$$

We take this to be an inductive definition of the set of consequences. That is, we are characterizing the smallest set of judgements $\Gamma \vdash a : T$ which obey the premises. In this way the inference rule (2) has the force of a biconditional corresponding to clause 4 in (1).

What counts as an object may vary from agent to agent (particularly if agents are of different species). Different agents have what Barwise (1989) would call different *schemes of individuation*. There appears to be a complex relationship between the types that an agent is attuned to and the parts of the world which the agent will perceive as an object. We model this in part by allowing different type systems to have different objects. In addition we will make extensive use in our systems of a basic type *Ind* for "individual" which corresponds to Montague's notion of "entity". The type *Ind* might be thought of as modelling a large part of an agent's scheme of individuation in Barwise's sense. However, this clearly still leaves a great deal to be explained and we do this in the hope that exploring the nature of the type systems involved will ultimately give us more insight into how individuation is achieved.

1.4 Situation types

Kim continues her walk in the park. She sees a boy playing with a dog and notices that the boy gives the dog a hug. In perceiving this event she is aware that two individuals are involved and that there is a relation holding between them, namely hugging. She also perceives that the boy is hugging the dog and not the other way around. She sees that a certain action (hugging) is being performed by an agent (the boy) on a patient (the dog). This perception seems more complex than the classification of an individual object as a tree in the sense that it involves two individual participants and a relation between them as well as the roles those two individuals play in the relation. While it is undoubtably more complex than the simple classification of an object as a tree, we want to say that it is still the assignment of a type to an object. The object is now an event and she classifies the event as a hugging event with the boy as agent and the dog as patient. We shall have complex types which can be assigned to such events.

Complex types are constructed out of other entities in the theory. As we have just seen, cognitive agents, in addition to being able to assign types to individual objects like trees, also perceive the world in terms of states and events where objects have properties and stand in relations to each other – what Davidson (1967) called events and Barwise and Perry (1983) called situations.

1.4.1 Types constructed from predicates (ptypes)

We introduce types which are constructed from predicates (like 'hug') and objects which are arguments to this predicate like a and b. We will represent such a constructed type as hug(a,b) and we will call it a *ptype* to indicate that it is a type whose main constructor is a predicate. What would an object belonging to such a type be? According to the type-theoretic approach introduced by Martin-Löf it should be an object which constitutes a proof that a is hugging b. For Martin-Löf, who was considering mathematical predicates, such proof objects might be numbers with certain properties, ordered pairs and so on. Ranta (1994) points out that for non-mathematical predicates the objects could be events as conceived by Davidson (1967, 1980). Thus hug(a,b) can be considered to be an event or a situation type. In some versions of situation theory Barwise (1989); Seligman and Moss (1997), objects (called *infons*) constructed from a relation and its arguments was considered to be one kind of situation type. Thus one view would be that ptypes are playing a similar role in type theory to the role that infons play in situation theory.

What kind of entity are predicates? One important fact about predicates is that they come along with an *arity*. The arity of a predicate tells you what kind of arguments the predicate takes and what order they come in. For us the arity of a predicate will be a sequence of types. The predicate 'hug' as discussed above we can think of as a two-place predicate both of whose arguments must be of type *Ind*, that is, an individual. Thus the arity of 'hug' will be $\langle$*Ind*, *Ind*$\rangle$. The idea is that if you combine a predicate with arguments of the appropriate types in the appropriate order indicated by the arity then you will have a type. Thus if a : *Ind* and b : *Ind* then hug(a,b) will be a type, intuitively the type of situation where a hugs b.

We will introduce a function *Arity* which is defined on predicates and which assigns an arity to any predicate. This function is introduced in a *predicate signature* which in addition tells you what predicates there are and what we can use to characterize their arguments (a set of types in the way we will use this. We define a predicate signature by the definition in (3) (repeated in Appendix 3.1).

(3) A *predicate signature* is a triple

$$\langle \mathbf{Pred}, \mathbf{ArgIndices}, \mathit{Arity} \rangle$$

where:

1. **Pred** is a set (of predicates)
2. **ArgIndices** is a set (of indices for predicate arguments, normally types)
3. *Arity* is a function with domain **Pred** and range included in the set of finite sequences of members of **ArgIndices**.

A simple example of a predicate signature would be given by (4).

(4) a. **Pred** = {boy, dog, hug}

b. **ArgIndices** = {*Ind*}

c. *Arity* is defined by:

$\mathit{Arity}(\text{boy}) = \langle \mathit{Ind} \rangle$
$\mathit{Arity}(\text{dog}) = \langle \mathit{Ind} \rangle$
$\mathit{Arity}(\text{hug}) = \langle \mathit{Ind}, \mathit{Ind} \rangle$

It may be desirable to allow some predicates to combine with more than one assortment of argument types. Thus, for example, one might wish to say that the predicate 'believe' can combine with two individuals just like 'hug' (as in *Kim believes Sam*) or with an individual and a "proposition" (as in *Kim believes that Sam is telling the truth*). Similarly the predicate 'want' might be both a two-place predicate for individuals (as in *Kim wants the tree*) or a two-place predicate between individuals and "properties" (as in *Kim wants to own the tree*). We shall have more to say about "propositions" and "properties" later. For now, we just note that we want to allow for the possibilities that predicates can be *polymorphic* in the sense that there may be more than one sequence of types which characterize the arguments they are allowed to combine with. The sequences need not even be of the same length (consider *Kim walked* and *Kim walked the dog*). We thus allow for the possibility that these pairs of natural language examples can be treated using the same polymorphic predicate. Another possibility, of course, is to say that the English verbs can correspond to different (though related) predicates in the example pairs and not allow this kind of predicate polymorphism in the type theory. We do not take a stand on this issue but merely note that both possibilities are available. If predicates are to be considered polymorphic then the arity of a predicate can be considered to be a set of sequences of types. In (5) we give a definition of a polymorphic predicate signature (repeated in Appendix 3.1).

(5) A *polymorphic predicate signature* is a triple

$$\langle \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity} \rangle$$

where:

1. **Pred** is a set (of predicates)
2. **ArgIndices** is a set (of indices for predicate arguments, normally types)
3. *Arity* is a function with domain **Pred** and range included in the powerset of the set of finite sequences of members of **ArgIndices**.

A simple example of a polymorphic predicate signature is given in (6).

(6) a. **Pred** = {boy, dog, hug, walk}

b. **ArgIndices** = $\{Ind\}$

c. *Arity* is defined by:

$$\begin{aligned}&\textit{Arity}(\text{boy}) = \{\langle \textit{Ind} \rangle\}\\ &\textit{Arity}(\text{dog}) = \{\langle \textit{Ind} \rangle\}\\ &\textit{Arity}(\text{hug}) = \{\langle \textit{Ind}, \textit{Ind} \rangle\}\\ &\textit{Arity}(\text{walk}) = \{\langle \textit{Ind} \rangle, \langle \textit{Ind}, \textit{Ind} \rangle\}\end{aligned}$$

An alternative to our characterization of predicates is to consider them as functions from sequences of objects matching their arity to types. As such they would be a *dependent type*, that is, an entity which returns a type when provided with an appropriate object or sequence of objects. However, we have not done this because we want all those entities we call dependent types to be representable as λ-expressions. We can, however, think of them as *type constructors* as will be made clear in our discussion of systems of complex types below.

A *system of complex types* adds to a system of basic types a collection of types constructed from a set of predicates with their arities, that is, it adds all the types which you can construct from the predicates by combining them with objects of the types corresponding to their arities according to the types in the rest of the system. The system also assigns a set of objects to all the types thus constructed from predicates. Many of these types will be assigned the empty set. Intuitively, if we have a type hug(c,d) and there are no situations in which c hugs d then there will be nothing in the extension of hug(c,d), that is, it will be assigned the empty set in the system of complex types. Notice that the intensionality of our type system becomes very important here. There may be many individuals x and y for which hug(x,y) is empty but still we would want to say that the types resulting from the combination of 'hug' with the various different individuals corresponds to different types of situations. The formal characterization of a system of complex types is given in (7) (repeated in Appendix 3.2).

(7) A *system of complex types* is a quadruple:

$$\textbf{TYPE}_C = \langle \textbf{Type}, \textbf{BType}, \langle \textbf{PType}, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \langle A, F\rangle\rangle$$

where:

1. $\langle$**BType**, $A\rangle$ is a system of basic types
2. **BType**$\subseteq$**Type**
3. for any $T \in$ **Type**, if $a :_{\langle \mathbf{BType}, A\rangle} T$ then $a :_{\mathbf{TYPE_C}} T$
4. $\langle$**Pred**, **ArgIndices**, *Arity*$\rangle$ is a (polymorphic) predicate signature
5. $P(a_1, \ldots a_n) \in$ **PType** iff $P \in$ **Pred**, $T_1 \in \mathbf{Type}, \ldots, T_n \in \mathbf{Type}$, *Arity*($P$)=$\langle T_1, \ldots, T_n\rangle$ ($\langle T_1, \ldots, T_n\rangle \in$*Arity*($P$)) and $a_1 :_{\mathbf{TYPE_C}} T_1, \ldots, a_n :_{\mathbf{TYPE_C}} T_n$
6. **PType**$\subseteq$**Type**
7. for any $T \in$ **PType**, $F(T)$ is a set disjoint from **Type**
8. for any $T \in$ **PType**, $a :_{\mathbf{TYPE_C}} T$ iff $a \in F(T)$

(7) perhaps looks a little forbidding for something that says that if you have a predicate P whose arity is $\langle T_1, \ldots, T_n\rangle$ and you have objects $a_1 : T_1, \ldots, a_n : T_n$ then $P(a_1, \ldots, a_n)$ is a ptype and any ptype is also a type. In this definition we have not made explicit exactly what set theoretic object we are representing with $P(a_1, \ldots, a_n)$. We will take this up below (in Section 1.4.2) since it is part of a general strategy we employ for representing entities in our type theory as sets. (7) also gives us a function F which maps ptypes to a set of witnesses and it also makes clear that a system of complex types adds to a system of basic types. The set of types of the new system consists of the basic types and the ptypes. Perhaps the informal proof theoretic notation in (8) is a little less forbidding.

(8) For Γ a system of complex types:

a. $$\frac{\Gamma \vdash T \in \textbf{BType} \qquad \Gamma \vdash a \in A(T)}{\Gamma \vdash a : T}$$

b. $$\frac{\Gamma \vdash T \in \textbf{BType}}{\Gamma \vdash T \in \textbf{Type}}$$

c. $$\frac{\Gamma \vdash P \in \textbf{Pred} \qquad \Gamma \vdash \langle T_1, \ldots, T_n \rangle = \mathit{Arity}(P) \qquad \Gamma \vdash a_1 : T_1, \ldots, \Gamma \vdash a_n : T_n}{\Gamma \vdash P(a_1, \ldots, a_n) \in \textbf{PType}}$$

or alternatively if we are considering our predicates to be polymorphic:

$$\frac{\Gamma \vdash P \in \textbf{Pred} \qquad \Gamma \vdash \langle T_1, \ldots, T_n \rangle \in \mathit{Arity}(P) \qquad \Gamma \vdash a_1 : T_1, \ldots, \Gamma \vdash a_n : T_n}{\Gamma \vdash P(a_1, \ldots, a_n) \in \textbf{PType}}$$

d. $$\frac{\Gamma \vdash T \in \textbf{PType}}{\Gamma \vdash T \in \textbf{Type}}$$

e. $$\frac{\Gamma \vdash T \in \textbf{PType} \qquad \Gamma \vdash s \in F(T)}{\Gamma \vdash s : T}$$

(8a) is the rule that we had for basic type systems except that we have identified the relevant set of types as **BType** (the basic types). (8b) tells us that **BType** is a subset of **Type**. (8c) tells us how to form ptypes from a predicate and an appropriate sequence of objects. (8d) tells us that ptypes are types. (8e) tells us that the witnesses of ptypes are determined by the function F.

There are thus two important functions in a system of complex types: one, which we call A, which comes from the system of basic types embedded in the system and assigns extensions to basic types and the other, which we call F, which assigns extensions to types constructed from predicates and arguments corresponding to the arity of the predicates. We have chosen the letters A and F because they are used very often in the characterization of models of first order logic. A model for first order logic is often characterized as a pair $\langle A, F \rangle$ where A is the domain and F a function which assigns denotations to the basic expressions (constants and predicates) of the logic. In a slight variation on classical first order logic A may be a sorted domain, that is the domain is not a single set but a set divided into various subsets, corresponding to *sorts*. For us, A characterizes assignments to basic types and thus provides something like a sorted domain in first order model theory. In first order logic F gives us what we need to know to determine the truth of expressions like 'hug(a,b)' in first order logic. Thus F will assign to the predicate 'hug' a set of ordered pairs telling us who hugs whom. Our F also give us the information we need in order to tell who stands in a predicate relation. However, it does this, not by assigning a set of ordered n-tuples to each predicate, but by assigning sets of witnesses (or "proofs") to each type constructed from a predicate with appropriate arguments. The set of ordered pairs assigned to 'hug' by the first order logic F corresponds to the set of pairs of arguments $\langle x, y \rangle$ for which the F in a complex system of types assigns a non-empty set. For this reason we call the pair $\langle A, F \rangle$ a *model* within the type system, even though it is not technically a model in the sense of model theory for logic. The correspondence becomes important later in the book, when we talk about modal type systems.

What are the entities which are witnesses for ptypes? The intuition is that, for example, (9) means that e is an event or situation where the individual a is running.

(9) e : run(a)

There are two competing intuitions about what e could be. One is that it is a “part of the world”, a non-set (urelement). That is, from the perspective of set theory and the theory of types it is an unstructured atom. The other intuition we have is that it is a structured entity which contains a as a component and in which a running activity is going on which involves smaller events such as picking feet up off the ground, spending certain time in each step cycle with neither foot touching the ground and so on. We want to allow for both of these intuitions. That is, a witness for a ptype can be a non-set corresponding to our notion of an event of a certain type. Or it can be the kind of labelled set which we call a record (see Section 1.4.3). That is, e is not only a witness for the type ‘run(a)’ but also for a record type which characterizes in more detail the structure of the event. We will argue that both intuitions are important and that observers of the world shift between views where certain ptypes are regarded as types of non-sets and views where those ptypes are types of records.

The introduction of predicates and ptypes raises the question of how one-place predicates relate to basic types. For example, what is the relationship between a type *Dog* whose witnesses are dogs and a predicate ‘dog’ whose arity is $\langle$*Ind*$\rangle$. One way to relate the two is given in (10).

(10) a : *Dog* iff $\exists e\ e$: dog(a)

(10) says that something is of type *Dog* just in case there is a situation which shows it to fall under the predicate ‘dog’. In this book we will relate common nouns to predicates rather than basic types, in part because common nouns can sometimes have more than one argument and in part because we want to limit the number of basic types we use. If we need a type we can derive it from the predicate using something like (10).

1.4.2 Representing complex entities as labelled sets

When we characterized ptypes in Section 1.4.1, we did not make explicit exactly which set-theoretic entity we were representing by the notation for a ptype '$P(a_1,\ldots,a_n)$'. In general complex entities in our theory will be a particular kind of set.

We introduce a notion *labelled sets* to model our complex entities. We will assume that our set theory comes equipped with a set of *urelements* (entities which are not sets but which can be members of sets). We will assume that among the urelements is a countably infinite set which is designated as the set of labels. A *labelled set* (see also Appendix 1) is a set of ordered pairs whose first member is a label and whose second element is either an urelement which is not a label or a set (possibly a labelled set), such that no more than one ordered pair can contain any particular label as its first member. This means that a labelled set is the traditional set theoretic construction of an extensional function from a set of labels onto some set. Suppose that we have a set (11a) and that $\ell_0,\ell_1,\ell_2,\ell_3$ are labels. Then examples of labelled sets which are *labellings* of (11a) would be (11b and c).

(11) a. $\{a,b,c,d\}$
b. $\{\langle\ell_0,a\rangle,\langle\ell_1,b\rangle,\langle\ell_2,c\rangle,\langle\ell_3,d\rangle\}$
c. $\{\langle\ell_0,\{\langle\ell_0,a\rangle,\langle\ell_1,b\rangle\}\rangle,\langle\ell_2,c\rangle,\langle\ell_3,d\rangle\}$

We will also sometimes have need of adding flavours to our labelled sets when we need to model distinct objects which correspond to the same set of ordered pairs. We will assume that there is a finite or countably infinite set of flavours among the urelements and that this set is disjoint from the set of labels. A flavoured labelled set contains a single flavour, $\mathfrak{f}$, in addition to the ordered pairs. Thus the labelled sets in (12) are examples of labelled sets with flavour $\mathfrak{f}$.

(12) a. $\{\mathfrak{f},\langle\ell_0,a\rangle,\langle\ell_1,b\rangle,\langle\ell_2,c\rangle,\langle\ell_3,d\rangle\}$
b. $\{\mathfrak{f},\langle\ell_0,\{\langle\ell_0,a\rangle,\langle\ell_1,b\rangle\}\rangle,\langle\ell_2,c\rangle,\langle\ell_3,d\rangle\}$

We will refer to the first members of the pairs in a labelled set as *labels* used in the labelled set and we will refer to the second members of the ordered pairs as the *labelled elements* of the labelled set. If X is a labelled set, we will use $\mathrm{labels}(X)$ to represent the set of labels of X, that is, the left projection of X which means the set of objects which are first members of the set of ordered pairs which are members of X. Note that this means that if X is the labelled set (11c) or (12b), then $\mathrm{labels}(X)$ is $\{\ell_0,\ell_2,\ell_3\}$, that is, the set of those labels which occur at the topmost level of X, not including the set of labels that occur within a labelled set contained in X, which in this case would in addition include the label ℓ_1. If X is a labelled set and $\ell\in\mathrm{labels}(X)$ we will use $X.\ell$ to represent the entity labelled by ℓ. Thus if X is (12b), $X.\ell_0$ is a. We can also define the set of *paths* in labelled sets given by the definition in (13).

(13) If X is a labelled set, then

1. if $\ell \in \mathrm{labels}(X)$, then $\ell \in \mathrm{paths}(X)$
2. if $\ell \in \mathrm{labels}(X)$, $X.\ell$ is a labelled set and $\pi \in \mathrm{paths}(X.\ell)$, then $\ell.\pi \in \mathrm{paths}(X)$

By this definition the set of paths in (11c) is (14).

(14) $\{\ell_0, \ell_2, \ell_3, \ell_0.\ell_0, \ell_0.\ell_1\}$

Note also that by these definitions, if X is (11c), then $X.\ell_0.\ell_0$ is a, that is, we can use the dot notation to take us down to a value on any path in the labelled set.

The set of *total paths* in a labelled set, X, $\mathrm{tpaths}(X)$ is the set of paths, π, such that $\pi \in \mathrm{paths}(X)$ and $X.\pi$ is not a labelled set.

There are various ways in which labelled sets could be represented graphically. One way to represent the examples in (11b and c) would be as in (15).

(15) a.
$$\begin{array}{cccc} \ell_0 & \ell_1 & \ell_2 & \ell_3 \\ | & | & | & | \\ \{a, & b, & c, & d\} \end{array}$$

b.
$$\begin{array}{cccc} \multicolumn{2}{c}{\ell_0} & & \\ / & \backslash & & \\ \ell_0 & \ell_1 & \ell_2 & \ell_3 \\ | & | & | & | \\ \{a, & b, & c, & d\} \end{array}$$

Labelled sets where we identify particular distinguished labels will always give us enough structure to model the structured entities that we need and define operations on them as required by our theory of types.

The entity represented by $P(a_1, \ldots a_n)$ is the labelled set in (16) where 'pred', 'arg$_i$' are reserved labels (that is, not used except as required here).

(16) $\{\langle \mathrm{pred}, P\rangle, \langle \mathrm{arg}_1, a_1\rangle, \ldots, \langle \mathrm{arg}_n, a_n\rangle\}$

1.4.3 Record types

Kim sees a situation where a (the boy) hugs b (the dog) and perceives it to be of type 'hug(a,b)'. However, there are intuitively other types which she could assign to this situation other than the type of situation where a hugs b which is represented here. For example, a more general type, which would be useful in characterizing all situations where hugging is going on between any individuals, is that of "situation where one individual hugs another individual". Another type of situation she might use is that of "situation where a boy hugs a dog". This is a more specific type than "situation where one individual hugs another individual" but still does not tie us down to the specific individuals a and b as the type 'hug(a,b)' does.

There are at least two different ways in type theory to approach these more general types. One is to use Σ-*types* such as (17).

(17) a. Σx:*Ind*.Σy:*Ind*.hug(x,y)

b. Σx:*Boy*.Σy:*Dog*.hug(x,y)

We will use the notation $T(\!(x_1, \ldots, x_n)\!)$ to represent that the type T depends on $x_1, \ldots, x_n$. For example, the types 'hug(x,y)' represented within the expressions in (17) depend on x and y. In general Σx:T_1.$T_2(\!(x)\!)$ will have as witnesses any ordered pair the first member of which is a witness for T_1 and the second member of which is a witness for $T_2(\!(x)\!)$. Thus this type will be non-empty ("true") just in case there is something a of type T_1 such that there is something of type $T_2(\!(a)\!)$. This means that Σ-types correspond to existential quantification. A witness for (17a) would be $\langle a, \langle b, s\rangle\rangle$ where a:*Ind*, b:*Ind* and s:hug(a,b). If there is such a witness then some individual hugs another individual and conversely if some individual hugs another individual there will be a witness for this type. Σ-types are exploited for the semantics of natural language by Ranta (1994) among others.

Another approach to these more general types is to use *record types* such as (18a,b) or, as we will prefer given our decision in Section 1.4.1 to use ptypes constructed from predicates rather than types corresponding to common nouns, (18c).

(18) a. $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{y} & : & \textit{Ind} \\ \text{e} & : & \text{hug(x,y)} \end{bmatrix}$

b. $\begin{bmatrix} \text{x} & : & \textit{Boy} \\ \text{y} & : & \textit{Dog} \\ \text{e} & : & \text{hug(x,y)} \end{bmatrix}$

c. $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c}_1 & : & \text{boy(x)} \\ \text{y} & : & \textit{Ind} \\ \text{c}_2 & : & \text{dog(y)} \\ \text{e} & : & \text{hug(x,y)} \end{bmatrix}$

In TTR, record types are labelled sets. A first approximation to the labelled sets represented in (18) is given in (19). (In Section 1.4.3.1 we will introduce a complication in connection with the dependency represented by 'hug(x,y)', 'boy(x)' and 'dog(y)'.)

(19) a. $\{\langle \text{x}, \textit{Ind}\rangle, \langle \text{y}, \textit{Ind}\rangle, \langle \text{e}, \text{hug(x, y)}\rangle\}$

b. $\{\langle \text{x}, \textit{Boy}\rangle, \langle \text{y}, \textit{Dog}\rangle, \langle \text{e}, \text{hug(x, y)}\rangle\}$

c. $\{\langle \text{x}, \textit{Ind}\rangle, \langle \text{c}_1, \text{boy(x)}\rangle, \langle \text{y}, \textit{Ind}\rangle, \langle \text{c}_2, \text{dog(y)}\rangle, \langle \text{e}, \text{hug(x, y)}\rangle\}$

'x', 'y', 'c_1', 'c_2' and 'e' are particular labels. In record types, the ordered pairs whose first member is a label are called *fields*. Thus record types are sets of fields. We will give a precise characterization of which labelled sets are record types later.

The witnesses of record types are *records*. These are also labelled sets, consisting of ordered pairs which we will call fields of the record. However, in this case the fields consist of a label and an object belonging to a type, rather than a type, as in the fields of record types. A record, r, is a witness for a record type, T, just in case r contains fields with the same labels as those in T and the objects in the fields in r are of the type with the corresponding label in T. The record may contain additional fields with labels not mentioned in the record type with the restriction there can only be one field within the record with a particular label. Thus both (20a) and (20b) are records of type (18a).

(20) a. $\begin{bmatrix} \text{x} & = & a \\ \text{y} & = & b \\ \text{e} & = & s \end{bmatrix}$ where a:*Ind*, b:*Ind* and s:hug(a,b)

b. $\begin{bmatrix} \text{x} & = & c \\ \text{y} & = & d \\ \text{e} & = & s' \\ \text{z} & = & a' \\ \text{w} & = & a'' \end{bmatrix}$ where c:*Ind*, d:*Ind*, s':hug(c,d) and a' and a'' are objects of some type

Note that in our notation for records we have '=' between the two elements of the field whereas in record types we have ':'. Note also that when we have types constructed from predicates in our record types and the arguments are represented as labels as in (18a) this means that the type is *dependent* on what objects you choose for those labels in the object of the record type. Thus in (20a) the type of the object labelled 'e' is hug(a,b) whereas in (20b) the type is hug(c,d). Actually, the notation we are using here for the dependent types is a convenient simplification of what is needed as we will explain later.

Record types and Σ-types are very similar in an important respect. The type (18a) will be witnessed ("true") just in case there are individuals x and y such that x hugs y. Thus both record types and Σ-types can be used to model existential quantification. In fact record types and Σ-types are so similar that you would probably not want to have both kinds of types in a single system and we will not use Σ-types. We have chosen to use record types for a number of reasons:

fields are unordered The Σ-types in (21) are distinct, although there is an obvious equivalence which holds between them.

(21) a. Σx:*Ind*.Σy:*Ind*.hug(x,y)

b. Σy:*Ind*.Σx:*Ind*.hug(x,y)

They are not only distinct types but they also have distinct sets of witnesses. The object $\langle a, \langle b, s\rangle\rangle$ will be of type (21a) just in case $\langle b, \langle a, s\rangle\rangle$ is of type (21b). In contrast, since we are regarding record types (and records) as *sets* of fields, (22a,b) are variant notations for the same type.

(22) a. $\left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{y} & : & \textit{Ind} \\ \text{c} & : & \text{hug(x,y)} \end{array}\right]$

b. $\left[\begin{array}{lll} \text{y} & : & \textit{Ind} \\ \text{x} & : & \textit{Ind} \\ \text{c} & : & \text{hug(x,y)} \end{array}\right]$

labels Record types (and their witnesses) include labelled fields which can be used to access *components* of what is being modelled. Components of a record are defined as objects which occur in a record. (A precise definition will be given later.) This is useful, for example, when we want to analyze anaphoric phenomena in language where pronouns and other words refer back to parts of previous meanings in the discourse. They can also be exploited in other cases where we want to refer to components of utterances or their meanings as in clarification questions.

discourse representation The labels in record types can play the role of discourse referents in discourse representation structures (DRSs, Kamp and Reyle, 1993) and record types of the kind we are proposing can be used to model DRSs.

dialogue game boards Record types have been exploited to model dialogue game boards or information states (see in particular Ginzburg, 2012).

feature structures Record types can be used to model the kind of feature structures that linguists like to use (as, for example, in linguistic theories like Head Driven Phrase Structure Grammar, HPSG, Sag *et al.*, 2003). Here the labels in record types correspond to attributes in feature structures.

frames Record types can also be used to model something very like the kinds of frames discussed in frame semantics (Fillmore, 1982, 1985; Ruppenhofer *et al.*, 2006) or in the psychological literature (Barsalou, 1992b, 1999). The labels in record types correspond to roles (frame elements).

For discussion of some of the various uses to which record types can be put see Cooper (2005). We will take up all of the uses named here as we progress.

Another way of approaching more general types such as "situation where a boy hugs a dog" is to use *contexts* as used in type theory. If we wish to express that an inference from "x hugs y" to "x touches y" we might consider doing it as in (23)

(23) x : *Ind*, y : *Ind*, e : hug(x,y) $\vdash$ e : touch(x,y)

(23) means that in a context where x and y are individuals and e is a witness for the type 'hug(x,y)' then e is also a witness for the type 'touch(x,y)'. Contexts (as represented to the left of the turnstile ('$\vdash$') in (23)) are standardly thought of as sequences of judgements. They are not standardly thought of being objects which are witnesses for types in the type theory. However, as we develop our semantic theory in this book, we will want to think of contexts as objects belonging to a certain type and to give semantic analyses in terms of types of context. Records and record types will enable us to do this. Thus, for example, (18a) models the type of context represented to the left of the turnstile in (23). As in the comparison with Σ-types there is a difference in that the judgements in a standard type theory context are ordered whereas the fields in a record type are unordered. This means that technically (24) is a distinct context from that in (23) even though there is an obvious equivalence between them.

(24) y : *Ind*, x : *Ind*, e : hug(x,y)

They correspond to the same record type, however.

Thus we use record types to replace both the Σ-types and contexts that one often finds in standard versions of type theory.

1.4.3.1 Dependent fields in record types

Consider again the record type (18c) repeated as (25).

(25) $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c}_1 & : & \text{boy(x)} \\ \text{y} & : & \textit{Ind} \\ \text{c}_2 & : & \text{dog(y)} \\ \text{e} & : & \text{hug(x,y)} \end{bmatrix}$

Strictly speaking the notations 'boy(x)', 'dog(y)' and 'hug(x,y)' do not represent ptypes as we have defined them since 'x' and 'y' are labels, not objects of type *Ind* as required by the arities of the predicates. What we mean by this notation is that the labels are to be replaced by whatever is in the field with that label in the record that we are checking against the type. Thus, for example, if we are checking whether the record in (26a) is of the type (25) we need to check that the judgements listed in (26b) are correct.

(26) a. $\begin{bmatrix} \text{x} & = & a \\ \text{c}_1 & = & s_1 \\ \text{y} & = & b \\ \text{c}_2 & = & s_2 \\ \text{e} & = & s_3 \end{bmatrix}$

b. a : *Ind*
s_1 : boy(a)
b : *Ind*
s_2 : dog(b)
s_3 : hug(a,b)

The notation 'boy(x)' in (25) thus actually encodes two pieces of information: firstly that we have what is known as a *dependent type*, a function which takes a certain type of object and returns a type, and secondly that we give an address where we should look for the object in the record that we are checking. We will represent functions using λ-expressions from a variant of the λ-calculus. The relevant functions for (25) are given in (27).

(27) a. λv:*Ind* . boy(v)
b. λv:*Ind* . dog(v)
c. λv_1:*Ind* . λv_2:*Ind* . hug(v_1,v_2)

We shall normally use v, v_1, v_2, ... for the variables in our functions. In general if ξ is a variable and $\varphi(\!(\xi)\!)$ represents an object containing the value of ξ, then we use the notation $\lambda\xi{:}T$. $\varphi(\!(\xi)\!)$

to represent the total function f whose domain is the set of witnesses of the type T and for any $a : T$, $f(a) = \varphi(\!(a)\!)$. We shall say something more precise about functions in Section 1.4.3.2.

The second piece of information we need to provide is where to find the object(s) which will serve as the arguments to these functions. This will be a sequence of *paths* in the record, providing a path for each argument to the function. A path is a string of labels separated by '.' and corresponds to the general notion of paths for labelled sets with the important exception that we exclude paths containing the distinguished labels used in non-record structures. We will make this notion precise in Section 1.4.3.4. In the case of our current example we only need paths consisting of one label.

We will represent the dependent field as containing an ordered pair consisting of the dependent type and the sequence of paths. Thus (25) is more explicitly represented as (28).

$$(28)\quad \begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c}_1 & : & \langle \lambda v{:}\mathit{Ind}\ .\ \mathrm{boy}(v),\ \langle \mathrm{x} \rangle \rangle \\ \mathrm{y} & : & \mathit{Ind} \\ \mathrm{c}_2 & : & \langle \lambda v{:}\mathit{Ind}\ .\ \mathrm{dog}(v),\ \langle \mathrm{y} \rangle \rangle \\ \mathrm{e} & : & \langle \lambda v_1{:}\mathit{Ind}\ .\ \lambda v_2{:}\mathit{Ind}\ .\ \mathrm{hug}(v_1,v_2),\ \langle \mathrm{x},\mathrm{y} \rangle \rangle \end{bmatrix}$$

This will be our "official" notation although we will continue to use the notation as in (25) for the sake of readability when it is not important to make this explicit.

The nature of dependent fields in record types as we have explained it here means that before we can give an explicit account of record types, we must first introduce type systems which contain functions and ensure that those functions can return types in order to give us the dependent types that we need. This we will do in Sections 1.4.3.2 and 1.4.3.3.

1.4.3.2 Functions and function types

In Cooper (2012b) we left it open exactly what kind of object a function is and assumed there was some theory of functions which would allow us to characterize them in terms of their domain and range. One option commonly used in a classical set theoretic setting is to let functions be modelled as their *graphs*, that is, a set of ordered pairs. The graph of a function f can be characterized as the set in (29).

$$(29)\quad \{\langle x, y\rangle \mid f(x) = y\}$$

Ideally, we want a notion of function that is more like a program or a procedure. That is, functions can be intensional in the sense that two distinct functions can correspond to the same graph. However, it seems that for the purposes at hand the standard extensional notion of function as a set of ordered pairs is sufficient and consistent with the fact that we want the λ-expressions we are using to represent unique functions. For this reason, we will model functions here as sets of ordered pairs in the classical set-theoretic way. Ultimately, we suspect that a more computational and intensional notion of function should be substituted.

In (30) we characterize a system of complex types with function types (repeated in Appendix 4).

(30) A system of complex types **TYPE**$_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ *has function types* if

1. for any $T_1, T_2 \in$ **Type**, $(T_1 \rightarrow T_2) \in$ **Type**
2. for any $T_1, T_2 \in$ **Type**, $f :_{\mathbf{TYPE_C}} (T_1 \rightarrow T_2)$ iff f is a function whose domain is $\{a \mid a :_{\mathbf{TYPE_C}} T_1\}$ and whose range is included in $\{a \mid a :_{\mathbf{TYPE_C}} T_2\}$

We specify a function type $(T_1 \rightarrow T_2)$ to be the labelled set (31) where 'dmn' ("domain") and 'rng' ("range") are reserved labels.

(31) $\{\langle \mathrm{dmn}, T_1\rangle, \langle \mathrm{rng}, T_2\rangle\}$

The choice of modelling functions as sets of ordered pairs means that $f :_{\mathbf{TYPE_C}} (T_1 \rightarrow T_2)$ just in case $f \subseteq \{a \mid a :_{\mathbf{TYPE_C}} T_1\} \times \{a \mid a :_{\mathbf{TYPE_C}} T_2\}$ and if $b \in \{a \mid a :_{\mathbf{TYPE_C}} T_1\}$ then there is exactly one c, such that $\langle b, c\rangle \in f$. We shall say that in this case the result of applying the function f to b, in symbols, $f(b)$, is c.

The informal proof theory version of (30) is given in (32).

(32) For Γ, a system of complex types with function types:

a. $$\frac{\Gamma \vdash T_1 \in \textbf{Type} \qquad \Gamma \vdash T_2 \in \textbf{Type}}{\Gamma \vdash (T_1 \rightarrow T_2) \in \textbf{Type}}$$

b. $$\frac{\begin{array}{cc} [\Gamma \vdash a : T_1] & [\Gamma \vdash f(a) : T_2] \\ \vdots & \vdots \\ \Gamma \vdash f(a) : T_2 & \Gamma \vdash a : T_1 \end{array}}{\Gamma \vdash f : (T_1 \rightarrow T_2)}$$

c. $$\frac{\Gamma \vdash f : (T_1 \rightarrow T_2) \qquad \Gamma \vdash a : T_1}{\Gamma \vdash f(a) : T_2}$$

d. $$\frac{\Gamma \vdash f : (T_1 \rightarrow T_2) \qquad \Gamma \vdash f(a) : T_2}{\Gamma \vdash a : T_1}$$

(32a) tells us that for any two types, T_1 and T_2, we can form the function type $(T_1 \rightarrow T_2)$. (32b) tells us that if we can prove that $f(a) : T_2$ from the assumption that $a : T_1$ and we can also prove from the assumption $f(a) : T_2$ that $a : T_1$, then $f : (T_1 \rightarrow T_2)$. The first premise requires that the function is defined on all witnesses for T_1 and the second premise requires that anything on which the function is defined is a witness for T_1. Jointly they require that the domain of the function (the set of objects on which it is defined) is the set of witnesses for T_1. (32c) tells us that if we have a function of type $(T_1 \rightarrow T_2)$ and an object of type T_1 then the result of applying the function to that object will be of type T_2. Conversely, (32d) tells us that if we have a function of type $(T_1 \rightarrow T_2)$ and the result of applying it to some object is of type T_2, then that object must be of type T_1.

If a function, f, is of type $(T_1 \rightarrow T_2)$ we say that the *domain type* of f, $\mathrm{domtype}(f)$, is T_1.

We introduce a notation for functions based on the λ-calculus. The notation is characterized in (33) where v is a variable in our notation.

(33) $\lambda v{:}T \,.\, \varphi$ is that function f such that for any $a : T$, $f(a)$ (the result of applying f to a) is represented by $\varphi[v \leftarrow a]$ (the result of replacing any free occurrence of v in φ with a).

In our informal proof theoretic representation this can be expressed by (34) (where again $[v \leftarrow a]$ represents the replacement of any free occurrence of the variable v by a).

(34) a.
$$\frac{\begin{array}{c}[\Gamma \vdash a : T_1] \\ \vdots \\ \Gamma \vdash \varphi[v \leftarrow a] : T_2\end{array}}{\Gamma \vdash \lambda v{:}T_1 \,.\, \varphi : (T_1 \rightarrow T_2)}$$

b.
$$\frac{\Gamma \vdash \lambda v{:}T_1 \,.\, \varphi : (T_1 \rightarrow T_2) \qquad \Gamma \vdash a : T_1}{\Gamma \vdash \lambda v{:}T_1 \,.\, \varphi\ (a) = \varphi[v \leftarrow a] : T_2}$$

In this section we have introduced functions from objects of one type to objects of another type. However, it does not yet give us the functions we need in the dependent fields of our record types since these are functions which take objects of a type and return a *type*, that is, they are *dependent types*.

1.4.3.3 The type *Type*

Up until now we have said that the witnesses of any type do not overlap with the set of types. We are now going to relax this requirement in a restricted way by introducing a special type called *Type* whose witnesses are any type, that is, members of the set **Type**. We will call type systems that have types of types in this way *intensional* since this will be a key feature of our treatment of natural language intensional constructions in Chapter 6.[1] Since *Type* is itself a type it will also be a member of the set **Type** and this will mean that it has itself as a witness, that is, *Type* : *Type*. For everyday working purposes we will assume that this is the system we have and ignore the fact that this is bringing us into danger of introducing Russell's paradox. In the remainder of this subsection we will show how the paradox can be avoided by using a technique called stratification. We will in future assume that our type systems are stratified in this way without mentioning it explicitly for the most part. If you are not interested in the details of this you can skip the rest of this subsection and come back to it if you feel the need.

Allowing types to belong to themselves puts us in danger of creating a situation in which Russell's paradox arises. If some members of **Type** belong to themselves then we should be able to talk of the set of types which do not belong to themselves, $\{T \in \mathbf{Type} \mid T \not: T\}$. Suppose that some model assigns this set to T'. Then the question arises whether T' belongs to itself and we can show that if $T' : T'$ then $T' \not: T'$ and if $T' \not: T'$ then $T' : T'$. In order to avoid this problem we will *stratify* (or *ramify*) our type system by introducing types of different *orders*. (For a discussion of stratification see Turner, 2005.) A type system of order 0 will be a system of complex types in the way we have defined it. The set of types, $\mathbf{Type}^1$ of a type system of order 1 based on this system will contain in addition to everything in the original type system a type, $Type^1$, to which all the types of order 0, members of the set $\mathbf{Type}^0$, belong. In general for all the natural numbers n, $Type^{n+1}$ will be a type to which all the types in $\mathbf{Type}^n$ belong.

We characterize an intensional system of complex types in (35) (repeated in Appendix 10).

(35) An *intensional system of complex types* is a family of quadruples indexed by the natural numbers:

$$\mathbf{TYPE}_{IC} = \langle \mathbf{Type}^n, \mathbf{BType}, \langle \mathbf{PType}^n, \mathbf{Pred}, \mathbf{ArgIndices}, \mathit{Arity}\rangle, \langle A, F^n\rangle\rangle_{n\in Nat}$$

where (using $\mathbf{TYPE}_{IC_n}$ to refer to the quadruple indexed by n):

1. for each n,$\langle \mathbf{Type}^n, \mathbf{BType}, \langle \mathbf{PType}^n, \mathbf{Pred}, \mathbf{ArgIndices}, \mathit{Arity}\rangle, \langle A, F^n\rangle\rangle$ is a system of complex types

[1] In Martin-Löf type theory, types of types are called *universes*. This is, however, potentially a confusing terminology for a theory relating to the kind of model theory which has been used in linguistics where "universe" has a different meaning.

2. for each n, $\mathbf{Type}^n \subseteq \mathbf{Type}^{n+1}$ and $\mathbf{PType}^n \subseteq \mathbf{PType}^{n+1}$
3. for each n, if $T \in \mathbf{PType}^n$ then $F^n(T) \subseteq F^{n+1}(T)$
4. for each $n > 0$, $\mathit{Type}^n \in \mathbf{Type}^n$
5. for each $n > 0$, $T :_{\mathbf{TYPE}_{ICn}} \mathit{Type}^n$ iff $T \in \mathbf{Type}^{n-1}$

Here, but not in Cooper (2012b), we make explicit that *Type* is a distinguished urelement and that *Type*^{n} represents the labelled set $\{\langle \mathrm{ord}, n\rangle, \langle \mathrm{typ}, \mathit{Type}\rangle\}$ where 'ord' and 'typ' are reserved labels ("order", "type").

In our informal proof theoretic notation we can characterize intensional systems of complex types as in (36).

(36) For $\{\Gamma^n\}_{n \in Nat}$ an intensional system of complex types

a. $$\frac{\Gamma^n \vdash T \in \textbf{Type}^n}{\Gamma^{n+1} \vdash T \in \textbf{Type}^{n+1}}$$

b. $$\frac{\Gamma^n \vdash T \in \textbf{PType}^n}{\Gamma^{n+1} \vdash T \in \textbf{PType}^{n+1}}$$

c. $$\frac{\Gamma^n \vdash a : T}{\Gamma^{n+1} \vdash a : T}$$

d. $$\frac{}{\Gamma^n \vdash \mathit{Type}^n \in \textbf{Type}^n} \quad n > 0$$

e. $$\frac{\Gamma^n \vdash T \in \textbf{Type}^n}{\Gamma^{n+1} \vdash T : \mathit{Type}^{n+1}}$$

f. $$\frac{\Gamma^n \vdash T : \mathit{Type}^n}{\Gamma^{n-1} \vdash T \in \textbf{Type}^{n-1}} \quad n > 0$$

For the most part in the remainder of this book we will suppress the n-superscripts. This means that we will characterize a function such as (37a) as being of the type (37b) whereas in fact it is a witness for all the types characterized in (37c).

(37) a. $\lambda v{:}\mathit{Ind}\ .\ \mathrm{dog}(v)$

b. $(\mathit{Ind}{\rightarrow}\mathit{Type})$

c. $\{(\mathit{Ind}{\rightarrow}\mathit{Type}^n) \mid n > 0\}$

1.4.3.4 Definitions of records and record types

A record according to a set of labels $\mathcal{L}$ and a type system $\mathbb{T}$ is a finite labelled set whose labels are included in $\mathcal{L}$ and whose labelled elements are witnesses of some type according to $\mathbb{T}$. Records are characterized by the definition in (38).

(38) r is a *record according to a set of labels* $\mathcal{L}$*, a set of distinguished labels,* $\mathcal{D}$*, and a type system* $\mathbb{T}$ (Appendix 11.1) iff r is a finite labelled set (Appendix 1) whose labels are included in $\mathcal{L}$ but not in $\mathcal{D}$ and for any labelled element, v, in r, there is some type T such that $v :_{\mathbb{T}} T$.

In giving our informal proof theoretic characterization of the set of records we will use (39) to mean that r is a record according to type system Γ and set of labels $\mathcal{L}$.

(39) $\Gamma, \mathcal{L} \vdash r$ record

We give the characterization inductively in (40).

(40) For Γ a type system, $\mathcal{L}$ a set of labels and $\mathcal{D}$ a set of distinguished labels

a. $$\frac{}{\Gamma, \mathcal{L} \vdash \emptyset \text{ record}}$$

b. $$\frac{\Gamma \vdash a : T \qquad \Gamma, \mathcal{L} \vdash r \text{ record} \qquad \ell \in \mathcal{L} - (\mathcal{D} \cup \mathrm{labels}(r))}{\Gamma, \mathcal{L} \vdash r \cup \{\langle \ell, a \rangle\} \text{ record}}$$

If r is a record and $\langle \ell, v \rangle$ is in r, we call $\langle \ell, v \rangle$ a *field* of r, ℓ a *label* in r and v a *value* in r (the *value of* ℓ *in* r). We use $r.\ell$ to denote v.

Records, as labelled sets, will have paths as defined for labelled sets. However, we will want in addition a more restricted notion of path for records which excludes those paths which include the distinguished labels used in non-record structures which may be values in a record as characterized in (41), repeated in Appendix 11.2.

(41) If r is a record, then

1. if $\ell \in \mathrm{labels}(r)$, then $\ell \in \mathrm{paths}_{\text{rec}}(r)$
2. if $\ell \in \mathrm{labels}(r)$, $r.\ell$ is a record and $\pi \in \mathrm{paths}_{\text{rec}}(r.\ell)$, then $\ell.\pi \in \mathrm{paths}_{\text{rec}}(r)$

Similarly, $\mathrm{tpaths}_{\mathrm{rec}}(r)$ is the set of paths, π, such that $\pi \in \mathrm{paths}_{\mathrm{rec}}(r)$ and $r.\pi$ is not a record.

We will sometimes use 'paths' and 'tpaths' without the subscript for these more restricted notions when there is no risk for confusion.

We use a tabular format to represent records. A record as given in (42a) is displayed as (42b).

(42) a. $\{\langle\ell_1, v_1\rangle, \ldots, \langle\ell_n, v_n\rangle\}$

b. $\begin{bmatrix} \ell_1 & = & v_1 \\ & \vdots & \\ \ell_n & = & v_n \end{bmatrix}$

We now move to characterizing record types. Record types are, with two exceptions of distinguished non-complex record types, flavoured labelled sets using a flavour which we represent as 'RT', the record-type flavour. We first characterize type systems which have non-dependent record types (that is, record types which do not have dependent fields). Non-dependent record types are labelled sets whose labelled elements are types. We characterize a type system with complex types and non-dependent record types in (43) (repeated in Appendix 11.2).

(43) A system of complex types $\mathbf{TYPE}_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ *has (non-dependent) record types based on* $\langle\mathcal{L}, \mathbf{RType}\rangle$, where $\mathcal{L}$ is a countably infinite set (of labels) and **RType** $\subseteq$ **Type** if

1. $\mathit{Rec} \in \mathbf{RType}$
2. $r :_{\mathbf{TYPE}_C} \mathit{Rec}$ iff r is a record according to $\mathcal{L}$ and $\mathbf{TYPE}_C$.
3. $\mathit{ERec} \in \mathbf{RType}$
4. $r :_{\mathbf{TYPE}_C} \mathit{ERec}$ iff $r = \emptyset$
5. if $\ell \in \mathcal{L}$ and $T \in \mathbf{Type}$, then $\{\mathrm{RT}, \langle\ell, T\rangle\} \in \mathbf{RType}$.
6. $r :_{\mathbf{TYPE}_C} \{\mathrm{RT}, \langle\ell, T\rangle\}$ iff $r :_{\mathbf{TYPE}_C} \mathit{Rec}$, $\langle\ell, a\rangle \in r$ and $a :_{\mathbf{TYPE}_C} T$.
7. if $R \in \mathbf{RType} - \{\mathit{Rec}, \mathit{ERec}\}$, $\ell \in \mathcal{L}$, ℓ does not occur as a label in R (i.e. there is no field $\langle\ell', T'\rangle$ in R such that $\ell' = \ell$) and $T \in \mathbf{Type}$, then $R \cup \{\langle\ell, T\rangle\} \in \mathbf{RType}$.
8. $r :_{\mathbf{TYPE}_C} R \cup \{\langle\ell, T\rangle\}$ iff $r :_{\mathbf{TYPE}_C} R$, $\langle\ell, a\rangle \in r$ and $a :_{\mathbf{TYPE}_C} T$.

In this definition we introduced two distinguished non-complex types: *Rec*, the type of all records (clauses 1 and 2), and *ERec*, the type of the empty record (clauses 3 and 4). In clauses 5 and 6 we introduce records with a single field, $\langle\ell, T\rangle$, the type of records which contain a field with label

ℓ and an object of type T. Clauses 7 and 8 are recursion clauses that add a single field, $\langle \ell, T\rangle$, to any record type with at least one field (that is, those record types which are labelled sets). A record will be a witness for the new type if it is of the old type and it contains a field with the label ℓ and an object of type T.

In terms of our informal proof theoretic notation this can be expressed as (44).

(44) For Γ a system of complex types which has record types based on $\langle \mathcal{L}, \text{RType}\rangle$, $\mathcal{L}$ a countably infinite set of labels:

a. $\dfrac{}{\Gamma, \mathcal{L} \vdash \textit{Rec} \in \text{RType}}$

b. $\dfrac{\Gamma, \mathcal{L} \vdash r \text{ record}}{\Gamma, \mathcal{L} \vdash r : \textit{Rec}}$

c. $\dfrac{\Gamma, \mathcal{L} \vdash r : \textit{Rec}}{\Gamma, \mathcal{L} \vdash r \text{ record}}$

d. $\dfrac{}{\Gamma, \mathcal{L} \vdash \textit{ERec} \in \text{RType}}$

e. $\dfrac{}{\Gamma, \mathcal{L} \vdash \emptyset : \textit{ERec}}$

f. $\dfrac{\Gamma, \mathcal{L} \vdash r : \textit{ERec}}{\Gamma, \mathcal{L} \vdash r = \emptyset : \textit{ERec}}$

g. $\dfrac{\Gamma \vdash T \in \textbf{Type}}{\Gamma, \mathcal{L} \vdash T \in \textbf{Type}}$

h. $\dfrac{\Gamma \vdash a : T}{\Gamma, \mathcal{L} \vdash a : T}$

i. $\dfrac{\Gamma, \mathcal{L} \vdash T \in \text{RType}}{\Gamma, \mathcal{L} \vdash T \in \textbf{Type}}$

j. $\dfrac{\Gamma, \mathcal{L} \vdash T \in \textbf{Type} \qquad \ell \in \mathcal{L}}{\Gamma, \mathcal{L} \vdash \{\text{RT}, \langle \ell, T\rangle\} \in \text{RType}}$

k. $\dfrac{\Gamma, \mathcal{L} \vdash r : \textit{Rec} \qquad \Gamma, \mathcal{L} \vdash a : T \qquad \langle \ell, a\rangle \in r}{\Gamma, \mathcal{L} \vdash r : \{\text{RT}, \langle \ell, T\rangle\}}$

l. $\dfrac{\Gamma, \mathcal{L} \vdash T \in \textbf{Type} \qquad \Gamma, \mathcal{L} \vdash R \in \text{RType} - \{\textit{Rec}, \textit{ERec}\} \qquad \ell \in \mathcal{L} - \text{labels}(R)}{\Gamma, \mathcal{L} \vdash R \cup \{\langle \ell, T\rangle\} \in \text{RType}}$

m. $\dfrac{\Gamma, \mathcal{L} \vdash a : T \qquad \Gamma, \mathcal{L} \vdash R \in \text{RType} \qquad \Gamma, \mathcal{L} \vdash r : R \qquad \ell \in \mathcal{L} - \text{labels}(R) \qquad \langle \ell, a\rangle \in r}{\Gamma, \mathcal{L} \vdash r : R \cup \{\langle \ell, T\rangle\}}$

(44a) tells us that *Rec* is a distinguished record type (corresponding to (43), clause 1). (44b and c) tell us that r:*Rec* just in case r is a record (corresponding to (43), clause 2). (44d) introduces

ERec as a distinguished record type (corresponding to (43), clause 3) and (44e and f) tells us that the empty set is the only witness for *ERec* (corresponding to (43), clause 4). (44g and h) tell us respectively that anything which is a type according to the system is also a type according to the system and the set of labels and similarly that anything which is a witness for a type according to the system will be a witness for that type according to the system and the set of labels. (44i) requires that any record type is also a type according to the system (corresponding to the requirement **RType**$\subseteq$**Type**). (44j) introduces record types with one field.$\langle \ell, T \rangle$ (corresponding to (43), clause 5) and (44k) tells us that a record containing a field with label ℓ and an object of type T will be a witness for such a record type (corresponding to (43), clause 6). (44l and m) are inductive rules which, respectively, tell us that you can add a new field, $\langle \ell, T \rangle$, to a record type, R, provided ℓ is not already a label of R (corresponding to (43), clause 7) and that a record, r, is of the new type just in case $r : R$ and it contains a field $\langle \ell, a \rangle$ such that $a : T$ (corresponding to (43), clause 8).

The issue of what counts as a path in a record type is a little complex. As for records it is based on the definition of paths for labelled sets but excluding the distinguished labels used in non-record structures such as ptypes. However, we will make an exception to this when we introduce meet types since it will be important to include paths into meet types. We will discuss this when we introduce meet types in Chapter 2. We will also want paths to go into dependent fields which always return a record type such that for any appropriate arguments to the function in the dependent field the resulting record types will all have same paths. This will become important when we introduce relabelling for record types in Chapter 4. We will define paths for record types in terms of the paths of the records which are their witnesses. The relevant definitions are given in (45).

(45) For a record type, T,

a. $\pi \in \mathrm{paths}_{\text{rectype}}(T)$ iff for any $r : T$, $\pi \in \mathrm{paths}_{\text{rec}}(r)$

b. $\pi \in \mathrm{tpaths}_{\text{rectype}}(T)$ iff for any $r : T$, $\pi \in \mathrm{tpaths}_{\text{rec}}(r)$

In terms of our informal proof theoretic notation this can be expressed as (46).

(46) For a record type, T,

a. $$\frac{\begin{array}{c}[r : T] \\ \vdots \\ \pi \in \mathrm{paths}_{\text{rec}}(r)\end{array}}{\pi \in \mathrm{paths}_{\text{rectype}}(T)}$$

b. $$\frac{\begin{array}{c}[r : T]\\ \vdots \\ \pi \in \mathrm{tpaths}_{\text{rec}}(r)\end{array}}{\pi \in \mathrm{tpaths}_{\text{rectype}}(T)}$$

As with $\mathrm{paths}_{\text{rec}}$, we will often suppress the subscript on $\mathrm{paths}_{\text{rectype}}$ when there is no risk of confusion.

We can add dependent record types to systems which have non-dependent record types. In order to do this we need dependent types, that is, functions which return types and for this we need the type *Type* as introduced in intensional systems of complex types characterized in (35). We characterize an intensional system of types with (non-dependent) record types in (47), repeated in Appendix 11.2.

(47) An *intensional system of complex types* $\textbf{TYPE}_{IC}$ = $\langle$**Type**^{n}, **BType**, $\langle$**PType**^{n}, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F^n\rangle\rangle_{n\in Nat}$ *has (non-dependent) record types based on* $\langle \mathcal{L}, \mathbf{RType}^n\rangle_{n\in Nat}$ if for each n, $\langle$**Type**^{n}, **BType**, $\langle$**PType**^{n}, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F^n\rangle\rangle$ has record types based on $\langle \mathcal{L}, \mathbf{RType}^n\rangle$ and

1. for each n, $\mathbf{RType}^n \subseteq \mathbf{RType}^{n+1}$
2. for each $n > 0$, $\mathit{RecType}^n \in \mathbf{Type}^n$
3. for each $n > 0$, $T :_{\mathbf{TYPE}_{IC_n}} \mathit{RecType}^n$ iff $T \in \mathbf{RType}^{n-1}$

The definition in (47) requires that an intensional system with record types is a family of systems with record types, indexed by the natural numbers such that any record type in a system indexed by a natural number n will also be a record type in the system indexed by $n+1$. It also introduces as distinguished type *RecType*^{n} for each level n above 0 whose witnesses are the record types of the level $n-1$. In our informal proof-theoretic notation this can be expressed in as in (48).

(48) For $\{\Gamma^n\}_{n\in Nat}$ an intensional system of complex types with (non-dependent) record types

a. $$\frac{\Gamma^n \vdash T \in \mathbf{RType}^n}{\Gamma^{n+1} \vdash T \in \mathbf{RType}^{n+1}}$$

b. $$\frac{}{\Gamma^n \vdash \mathit{RecType}^n \in \mathbf{Type}^n}\; n > 0$$

c. $$\frac{\Gamma^n \vdash T \in \mathbf{RType}^n}{\Gamma^{n+1} \vdash T : \mathit{RecType}^{n+1}}$$

d. $$\frac{\Gamma^n \vdash T : \mathit{RecType}^n}{\Gamma^{n-1} \vdash T \in \mathbf{RType}^{n-1}}\; n > 0$$

(48a) requires that any record type on one level will be a record type on the next higher level (corresponding to (47), clause 1). (48b) introduces the distinguished type *RecType* on all levels above 0 (corresponding to (47), clause 2). (48c and d) requires that the witnesses of *RecType* are exactly the record types which are in the system one level down (corresponding to (47), clause 3).

Now we can introduce dependent record types by adding them to intensional type systems with record types. The characterization of type systems with dependent record types is given in (49) (repeated in Appendix 11.2).

(49) An *intensional system of complex types* **TYPE**$_{IC}$ = ⟨**Type**^{n}, **BType**, ⟨**PType**^{n}, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F^n\rangle\rangle_{n\in Nat}$ *has dependent record types based on* $\langle\mathcal{L}, \mathbf{RType}^n\rangle_{n\in Nat}$, if it has record types based on $\langle\mathcal{L}, \mathbf{RType}^n\rangle_{n\in Nat}$ and for each $n > 0$

1. if $R \in \mathbf{RType}^n$, $\ell \in \mathcal{L} - \text{labels}(R)$, $T_1, \ldots, T_m \in \mathbf{Type}^n$, $\pi_1, \ldots, \pi_m \in \text{paths}(R)$ and $\mathcal{T}$ is a function of type $(T_1 \to \ldots \to (T_m \to \mathit{Type}^n)\ldots)$, then $R \cup \{\langle\ell, \langle\mathcal{T}, \langle\pi_1, \ldots, \pi_m\rangle\rangle\rangle\} \in \mathbf{RType}^n$.
2. $r :_{\mathbf{TYPE}_{IC_n}} R \cup \{\langle\ell, \langle\mathcal{T}, \langle\pi_1, \ldots, \pi_m\rangle\rangle\rangle\}$ iff $r :_{\mathbf{TYPE}_{IC_n}} R$, $\langle\ell, a\rangle$ is a field in r, $r.\pi_1 :_{\mathbf{TYPE}_{IC_n}} T_1, \ldots, r.\pi_m :_{\mathbf{TYPE}_{IC_n}} T_m$ and $a :_{\mathbf{TYPE}_{IC_n}} \mathcal{T}(r.\pi_1)\ldots(r.\pi_m)$.

(49), clause 1 says that for any record type R we can create a new record type by adding a field $\langle\ell, \langle\mathcal{T}, \langle\pi_1, ..., \pi_m\rangle\rangle\rangle$, where ℓ is not a topmost label in R, $\mathcal{T}$ is an m-place function which returns a type and $\pi_1, ..., \pi_m$ are paths in R. Clause 2 says that a record r is of the new type just in case it is of type R, $\langle\ell, a\rangle$ is a field in r, $r.\pi_1, \ldots, r.\pi_m$ are appropriate arguments to $\mathcal{T}$ and a is of the type resulting from the application of $\mathcal{T}$ to $r.\pi_1, \ldots, r.\pi_m$. In terms of our informal proof theoretic notation we can express this as (50).

(50) For $\{\Gamma^n\}_{n\in Nat}$ an intensional system of complex types with dependent record types based on $\langle\mathcal{L}, \mathbf{RType}^n\rangle_{n\in Nat}$

a. $$\frac{\Gamma^n \vdash R \in \mathbf{RType}^n \quad \Gamma^n \vdash T_1, \ldots, T_m \in \mathbf{Type}^n \quad \ell \in \mathcal{L} - \text{labels}(R) \quad \Gamma^n \vdash \mathcal{T} : (T_1 \to \ldots \to (T_m \to \mathit{Type}^n)\ldots) \quad \pi_1, ..., \pi_m \in \text{paths}(R)}{\Gamma^n \vdash R \cup \{\langle\ell, \langle\mathcal{T}, \langle\pi_1, \ldots, \pi_m\rangle\rangle\rangle\} \in \mathbf{RType}^n}$$

b. $$\frac{\Gamma^n \vdash r : R \quad \langle\ell, a\rangle \in r \quad \ell \in \mathcal{L} - \text{labels}(R) \quad \Gamma^n \vdash a : \mathcal{T}(r.\pi_1)\ldots(r.\pi_m)}{\Gamma^n \vdash r : R \cup \{\langle\ell, \langle\mathcal{T}, \langle\pi_1, \ldots, \pi_m\rangle\rangle\rangle\}}$$

(50a) corresponds to (49), clause 1. It says that for any record type R not containing ℓ among its labels, any sequence of a subset of R's paths, $\pi_1, \ldots, \pi_m$, and a dependent type $\mathcal{T}$ with m

arguments, we can obtain a new record type by adding the field, $\langle \ell, \langle \mathcal{T}, \langle \pi_1, \ldots, \pi_m \rangle\rangle\rangle$ to R. (50b) corresponds to (49), clause 2. It says that if a record, r, is of type R and contains a field $\langle \ell, a\rangle$, where ℓ is not a label in R and a is of the type obtained by applying the dependent type $\mathcal{T}$ to $r.\pi_1, \ldots, r.\pi_m$, then r is of the record type resulting from adding the field $\langle \ell, \langle \mathcal{T}, \langle \pi_1, \ldots, \pi_m \rangle\rangle\rangle$ to R. As usual, the force of the biconditional in (49b) is provided by the fact that (50b) is part of an inductive definition and it is the only rule which specifies under which conditions a record is a witness for the record type with the additional dependent field.

Note that both records and record types may contain types in their fields. Minimal examples are given in (51).

(51) a. $\left[\begin{array}{lll} \ell & = & T \end{array}\right]$

b. $\left[\begin{array}{lll} \ell & : & T \end{array}\right]$

The record and the type in (51) are distinct objects which nevertheless seem to correspond to the same set of ordered pairs. We distinguish them by modelling the record as an unflavoured labelled set and the record type as a flavoured labelled set as given in (52a and b) respectively.

(52) a. $\{\langle \ell, T\rangle\}$

b. $\{\mathrm{RT}, \langle \ell, T\rangle\}$

1.4.3.5 Subtyping in record types

An important property of record types is that they introduce a restrictive notion of subtyping. Intuitively T_1 is a subtype of T_2 (in symbols $T_1 \sqsubseteq T_2$) just in case for any a, $a : T_1$ implies $a : T_2$, *no matter what is assigned to the basic types and ptypes*. Consider the types in (53).

(53) a. $\left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_1 & : & \mathrm{boy(x)} \\ \mathrm{y} & : & \textit{Ind} \\ \mathrm{c}_2 & : & \mathrm{dog(y)} \end{array}\right]$

b. $\left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_1 & : & \mathrm{boy(x)} \\ \mathrm{y} & : & \textit{Ind} \\ \mathrm{c}_2 & : & \mathrm{dog(y)} \\ \mathrm{e} & : & \mathrm{hug(x,y)} \end{array}\right]$

(53a) is intuitively the type of situation in which there is a boy and a dog. (53b) is the type of situation in which there is a boy and a dog and the boy hugs the dog. Clearly any situation of type (53b) must be of type (53a). This holds independently of what boys and dogs there are and what kind of hugging is going on. We can tell that (53b) is a subtype of (53a) simply by the fact that the set of fields of (53a) is a subset of the set of fields of (53b). We will notice other ways in which you can recognize that one record type is a subtype of another as we progress.

Subtyping on this view is a *modal* notion. This means that we do not just consider one type system but a collection of type systems which assign different objects to the basic types and ptypes. We call such a collection a *modal type system*. This corresponds intuitively to a set of possibilities under consideration. A type, T_1, is a subtype of another type, T_2, with respect to a modal type system just in case for each of the systems in the modal type system any witness for T_1 is also a witness for T_2. We characterize a modal system of complex types as a collection of systems of complex types in (54), repeated in Appendix 9. We use M as a variable over pairs $\langle A, F\rangle$ where A is an assignment of witnesses to the basic types of a system of complex types and F is an assignment to the ptypes of that system.

(54) A *modal system of complex types based on* $\mathcal{M}$ is a family of quadruples:

$$\textbf{TYPE}_{MC} = \langle \textbf{Type}_M, \textbf{BType}, \langle \textbf{PType}_M, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, M\rangle_{M\in\mathcal{M}}$$

where for each $M \in \mathcal{M}$, $\langle \textbf{Type}_M, \textbf{BType}, \langle \textbf{PType}_M, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, M\rangle$ is a system of complex types.

We call the individual systems of complex types in such a system the *possibilities* of the system. The set of basic types and the predicates and their arities are held constant across the different possibilities whereas the set of ptypes (and therefore the set of types in general) will vary since different assignments to the basic types will generate different arguments to the predicates and thus different ptypes. We can then define the notion of subtype with respect to a modal system of complex types as in (55).

(55) T_1 *is a subtype$_i$ of* T_2 *in* $\textbf{TYPE}_{MC}$, $T_1 \sqsubseteq_{\textbf{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$, if T_1 and T_2 are members of $\textbf{Type}_M$, then $\{a \mid a :_{\textbf{TYPE}_{MC_M}} T_1\} \subseteq \{a \mid a :_{\textbf{TYPE}_{MC_M}} T_2\}$

We can recast this in our informal proof theoretic notation as (56).

(56) For $\mathcal{G}$ a modal system of complex types

a.
$$\cfrac{\begin{array}{c}[\Gamma \in \mathcal{G},\ \Gamma \vdash x : T_1,\ \Gamma \vdash T_2 \in \textbf{Type}]\\ \vdots \\ \Gamma \vdash x : T_2\end{array}}{\mathcal{G} \vdash T_1 \sqsubseteq T_2}$$

b. $$\frac{\mathcal{G} \vdash T_1 \sqsubseteq T_2 \quad \Gamma \in \mathcal{G} \quad \Gamma \vdash a : T_1 \quad \Gamma \vdash T_2 \in \textbf{Type}}{\Gamma \vdash a : T_2}$$

A disadvantage of (56a) is that it does not tell us exactly what types would count as being in the subtype relation on the basis of the nature of the types. In the case of record types we can often recognize the subtype relation on the basis of the structure of the types. For example, a record type T_1 is a subtype of a record type T_2 if $T_2 \subseteq T_1$. (Recall that record types are sets of fields.) This means that we can have the informal inference rule in (57).

(57) For $\mathcal{G}$, a modal system of complex types

$$\frac{\Gamma, \Delta \in \mathcal{G} \quad \Gamma \vdash T_1 \in \textbf{RType} \quad \Delta \vdash T_2 \in \textbf{RType} \quad T_2 \subseteq T_1}{\mathcal{G} \vdash T_1 \sqsubseteq T_2}$$

We will introduce more such specific inference rules later.

1.5 Intensionality: propositions as types

Kim continues to think about the boy and the dog as she walks along. It was fun to see them playing together. They seemed so happy. The boy obviously thought that the dog was a good playmate. Kim is not only able to perceive events as being of certain types. She is able to recall and reflect on these types. She is able to form attitudes towards these types: it was fun that the boy and the dog were playing but a little worrying that they were so close to the pond. This means that the types themselves seem to be arguments to predicates like 'fun' and 'worrying'. This seems to be an important human ability – not only to be able to take part in or observe an event and find it fun or worrying but to be able to reflect independently of the actual occurrence of the event that it or in general similar events are fun or worrying. This is a source of great richness in human cognition in that it enables us to consider situation types independently of their actual instantiation.[2] This abstraction also enables us to consider what attitudes other individuals might have. For example, Kim believes that the boy thought that the dog was a good playmate. She is able to ascribe this belief to the boy. Furthermore, we are able to reflect on Kim's state of mind where she has a belief concerning the type of situation where the boy thinks that the dog was a good playmate. And somebody else could consider of us that we have a certain belief about Kim concerning her belief about the boy's belief. There is in principle no limit to the depth of recursion concerning our attitudes towards types.

[2]This richness also has its downside in that we often become so engaged in our internal cognitive abstraction that it can be difficult to be fully present and conscious of our direct perception of the world – for example, worrying about what might happen in the future rather than enjoying the present.

We propose to capture this reflective nature of human cognition by making the type theory technically *reflective* in the sense that we allow types themselves to be objects which can belong to other types. In classical model theoretic semantics we think of *believe* as corresponding to a relation between individuals and propositions. In our type theory, however, we are subscribing to the "propositions as types" view which comes to us via Martin-Löf (1984) and has its origins in intuitionistic logic (see Ranta, 1994, Section 2.16 for discussion). Propositions are true or false. Types of situations such as hug(a,b) correspond to propositions in the sense that if they are non-empty then the proposition is true. If there is nothing of this type then it is false. The reasoning is thus that we do not need propositions in our system as separate semantic objects if we already have types. We can use the types to play the role of propositions. To believe a type is to believe it to be non-empty. From the point of view of a type theory for cognition in which we connect types to our basic perceptual ability, this provides a welcome link between our perceptual ability and our ability to entertain propositions (that is, to consider whether they are true or false). We will develop this idea in Chapter 6.

1.6 Summary

In this chapter we started with a notion of perception as type assignment: perceiving an object or a situation involves judging it to be of a certain type. We ended with the promise that the kind of types used for perception will be used in the analysis of intensional constructions in natural language which involve attitudes (such as belief or desire) to propositions using the "propositions as types" dictum. An important point here is that while the origin of the system of types we use lies in perception, there is no claim that all the types which we can have attitudes to are types which have been used in perception. We have developed the ability to reflect on and reason about the types themselves even types that we have not encountered any witness for or that we know could not possibly have a witness.

In this chapter we introduced four kinds of types and a way of modelling them mathematically: basic types which are not structured, ptypes which are constructed from a predicate and appropriate arguments to the predicate, function types and record types containing labelled fields which contain types. Record types may contain dependent fields where the type in the field depends on objects in other fields. In the remainder of the book we will add more kinds of types as we need them.

Chapter 2

From event perception and action to information states and information exchange

2.1 Introduction

In Chapter 1 we talked about two kinds of situation types: ptypes and record types. This presents a static view of situations which are events, that is, those situations in which a change takes place. In Section 2.2 we will introduce string types which enable us to treat events as strings of smaller events.

2.2 The string theory of events

Kim stands and watches the boy and the dog for a while. They start to play fetch.[1] This is a moderately complex game in that it consists of a number of components which are carried out in a certain order. The boy picks up a stick attracts the attention of the dog (possibly shouting "Fetch!"), and throws the stick. The dog runs after the stick, picks it up in his mouth and brings it back to the boy. This sequence can be repeated arbitrarily many times. One thing that becomes clear from this is that events do not happen in a single moment but rather they are stretched out over intervals of time, characterized by the sub-events that constitute them. So if we were to have a type of event (that is, a type of situation) play_fetch(a,b,c) where a is a human, b is a dog and c is a stick we can say something about the series of subevents that we have identified. So we might draw an informal diagram something like Figure 2.1.

In an important series of papers including Fernando (2004, 2006, 2008, 2009, 2011, 2015), Fernando introduces a finite state approach to event analysis where events are analyzed in terms of

[1] `http://en.wikipedia.org/wiki/Fetch_(game)`, accessed 10th Oct 2011.

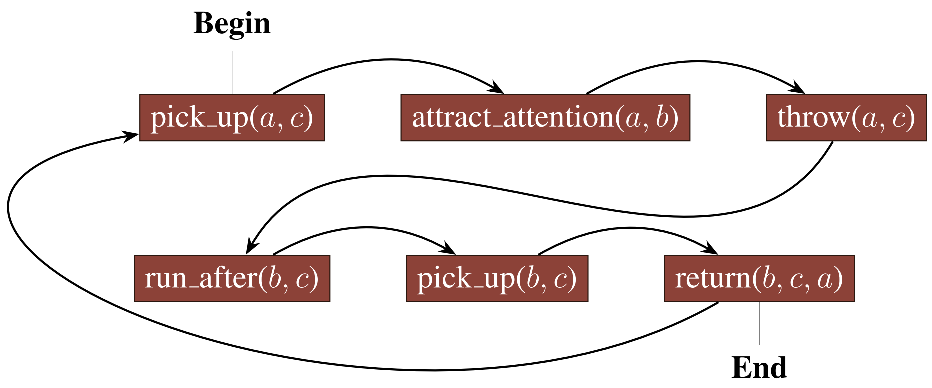


Figure 2.1: play_fetch(a,b,c)

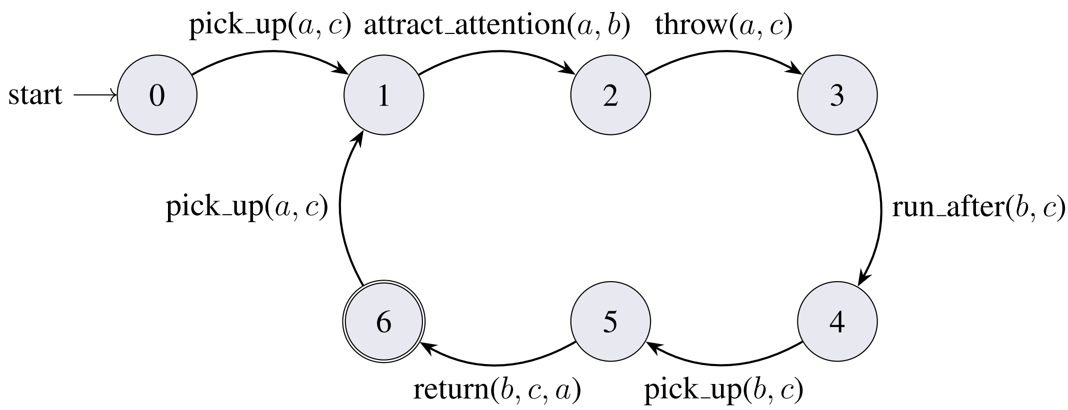


Figure 2.2: play_fetch(a,b,c) as a finite state machine

finite state automata something like what we have represented in Figure 2.2. Such an automaton will recognize a string of sub-events. The idea is that our perception of complex events can be seen as strings of punctual observations similar to the kind of sampling we are familiar with from audio technology and digitization processing in speech recognition. Thus events can be analyzed as strings of smaller events. Any object of any type can be part of a string. Any two objects (including strings themselves), s_1 and s_2, can be *concatenated* to form a string $s_1 s_2$. An important property of concatenation is *associativity*, that is if we concatenate s_1 with s_2 and then concatenate the result with s_3 we get the same string that we would obtain by concatenating s_2 with s_3 and then concatenating s_1 with the result. In symbols: $(s_1 s_2)s_3 = s_1(s_2 s_3)$. For this reason we normally write $s_1 s_2 s_3$ (without the parentheses). Following Fernando we will use these strings to give us our notion of temporal order.

If $a_1, a_2, \ldots, a_n$ are objects we will normally represent the string of these objects as $a_1 a_2 \ldots a_n$. Where confusion might arise from this notation we use $\mathrm{str}(a_1 a_2 \ldots a_n)$. This latter notation will be particularly useful when distinguishing a single object, a, from a unit string containing this object $\mathrm{str}(a)$. Although we will present strings in this way, we will model them as records with distinguished labels related to the natural numbers, t_0, t_1, $\ldots$ ('t' for "time"). The field labelled t_n will correspond to the $n+1$th place in the string. Thus a string of objects $a_1 a_2 a_3$ will be the record in (1).

(1) $\begin{bmatrix} \mathrm{t}_0 & = & a_1 \\ \mathrm{t}_1 & = & a_2 \\ \mathrm{t}_2 & = & a_3 \end{bmatrix}$

The concatenation of (1) with the string a_4, that is, (2a), will be (2b).

(2) a. $\begin{bmatrix} \mathrm{t}_0 & = & a_4 \end{bmatrix}$

b. $\begin{bmatrix} \mathrm{t}_0 & = & a_1 \\ \mathrm{t}_1 & = & a_2 \\ \mathrm{t}_2 & = & a_3 \\ \mathrm{t}_3 & = & a_4 \end{bmatrix}$

Strings can be introduced into a type system with record types by the definition in (3) (repeated in Appendix 19)

(3) A system of complex types $\mathbf{TYPE}_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ with record types based on $\langle \mathcal{L}, \mathbf{RType}\rangle$ *has strings* if

1. for each natural number i, $\mathrm{t}_i \in \mathcal{L}$
2. *String* $\in$ **BType**
3. $\emptyset :_{\mathbf{TYPE}_C}$ *String*
4. if $T \in$ **Type** and $a :_{\mathbf{TYPE}_C} T$ then $\{\langle \mathrm{t}_0, a\rangle\}$: *String*
5. if $s :_{\mathbf{TYPE}_C}$ *String*, $\mathrm{t}_n \in \mathrm{labels}(s)$ such that there is no $i > n$ where $\mathrm{t}_i \in \mathrm{labels}(s)$, $T \in$ **Type** and $a :_{\mathbf{TYPE}_C} T$ then $s \cup \{\langle \mathrm{t}_{n+1}, a\rangle\} :_{\mathbf{TYPE}_C}$ *String*
6. Nothing is of type *String* except as required above.

(3), clause 1 ensures that the labels t_i are among the labels available for forming record types. Clause 2 introduces a basic type *String*. Clause 3 says the empty set (also known as the empty record and empty string) is a string. Clause 4 says that if you have an object, a, of some type then you can form a unit string containing a. That is the record $\left[\mathrm{t}_0{=}a\right]$. Note that we always start with the label 't_0'. Clause 5 says that we can always create a new string from a string s by adding an additional object to the end of it, using a label 't_{n+1}' where n is the highest number such that $\mathrm{t}_n \in \mathrm{labels}(s)$. Clause 6 is an exclusion clause. Clauses 3–6 constitute an inductive definition of the set of witnesses of the type *String*.

In our informal proof theoretic notation this can be characterized by giving an inductive definition as in (4).

(4) For Γ a system of complex types with record types based on $\langle \mathcal{L}, \mathbf{RType} \rangle$ and strings

a. $\dfrac{}{\mathrm{t}_i \in \mathcal{L}}\ i \in \mathit{Nat}$

b. $\dfrac{}{\Gamma \vdash \mathit{String} \in \textbf{BType}}$

c. $\dfrac{}{\Gamma \vdash \emptyset : \mathit{String}}$

d. $\dfrac{\Gamma \vdash a : T}{\Gamma \vdash \{\langle \mathrm{t}_0, a \rangle\} : \mathit{String}}$

e. $\dfrac{\Gamma \vdash s : \mathit{String} \qquad \Gamma \vdash a : T \qquad \{\mathrm{t}_0, \ldots, \mathrm{t}_n\} = \mathrm{labels}(s)}{\Gamma \vdash s \cup \{\langle \mathrm{t}_{n+1}, a \rangle\} : \mathit{String}}\ n \geq 0$

We will continue to represent strings for convenience in the traditional way but modelling strings as records will become important when following paths in records down to elements in strings and any operations we define on records will automatically generalize to strings. We will use ε to represent the empty string (that is, the empty set). We will use $s[n]$ to represent the nth element in a string s (where the first element in the string is $s[0]$). In terms of the record notation this is just a convenient abbreviation for $s.\mathrm{t}_n$.

We will use $T^{=n}$, or, when there is no risk of confusion, simply T^n, as the type of strings of length n all of whose elements are of type T. We will use $T^{\geq n}$ for the type of strings of objects of type T which have length greater than or equal to n. In particular we will use T^* (the Kleene star) for $T^{\geq 0}$ and T^+ (the Kleene plus) for $T^{\geq 1}$.

We can make this precise with the definition in (5) (repeated in Appendix 19)

(5) A system of complex types **TYPE**$_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ with strings has *length determining string types* if

1. for any $T \in$ **Type** and n a natural number, the string types $T^{=n}$, $T^{\leq n}$, $T^{\geq n} \in$ **Type**
2. $s :_{\mathbf{TYPE}_C} T^{=n}$ $(T^{\leq n}, T^{\geq n})$ iff $s :_{\mathbf{TYPE}_C}$ *String*, for all i, $0 \leq i < \text{length}(s)$, $s[i] :_{\mathbf{TYPE_C}} T$ and $\text{length}(s) = (\leq, \geq)\ n$

In our informal proof theoretic notation this can be expressed as (6).

(6) For Γ a system of complex types with strings and length determining string types:

a. $\dfrac{\Gamma \vdash T \in \textbf{Type}}{\Gamma \vdash T^{=n} \in \textbf{Type}}\ n \in \textit{Nat}$ $\qquad \dfrac{\Gamma \vdash T \in \textbf{Type}}{\Gamma \vdash T^{\leq n} \in \textbf{Type}}\ n \in \textit{Nat}$

$\dfrac{\Gamma \vdash T \in \textbf{Type}}{\Gamma \vdash T^{\geq n} \in \textbf{Type}}\ n \in \textit{Nat}$

b. $$\dfrac{\Gamma \vdash s : \textit{String} \qquad \text{length}(s) = n \qquad \begin{array}{c}[i < n]\\ \vdots \\ \Gamma \vdash s[i] : T\end{array}}{\Gamma \vdash s : T^{=n}}$$

$$\dfrac{\Gamma \vdash s : \textit{String} \qquad \text{length}(s) \leq n \qquad \begin{array}{c}[i < \text{length}(s)]\\ \vdots \\ \Gamma \vdash s[i] : T\end{array}}{\Gamma \vdash s : T^{\leq n}}$$

$$\dfrac{\Gamma \vdash s : \textit{String} \qquad \text{length}(s) \geq n \qquad \begin{array}{c}[i < \text{length}(s)]\\ \vdots \\ \Gamma \vdash s[i] : T\end{array}}{\Gamma \vdash s : T^{\geq n}}$$

(6a) introduces the string types and (6b) specifies witness conditions for the types.

Next we introduce concatenation types. For any two types, T_1 and T_2, we can form the type $T_1{}^\frown T_2$. This is the type of strings ab where $a : T_1$ and $b : T_2$. The concatenation operation on types (just like that on objects) is associative so we do not use parentheses when more than one type is involved, e.g. $T_1{}^\frown T_2{}^\frown T_3$.

This can be made precise as (7), repeated in Appendix 19.

(7) A system of complex types **TYPE**$_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ with strings and length determining string types has *concatenation types* if

1. if $T_1, T_2 \in$ **Type** then the string type $T_1^\frown T_2 \in$ **Type**
2. $s :_{\mathbf{TYPE}_C} T_1^\frown T_2$ iff there are s_1 and s_2 such that
   a) $s_1 s_2 = s$
   b) $s_1 :_{\mathbf{TYPE}_C} T_1$ if T_1 is a string type, otherwise $s_1 :_{\mathbf{TYPE}_C} T_1^{=1}$
   c) $s_2 :_{\mathbf{TYPE}_C} T_2$ if T_2 is a string type, otherwise $s_2 :_{\mathbf{TYPE}_C} T_2^{=1}$

Clause 1 introduces concatenation types, $T_1^\frown T_2$, and clause 2 says that witnesses for concatenative types must be the concatenation of two strings which are of the types T_1 and T_2 resepctively if they are string types of if they are not they must be of the type of singleton strings whose only element is of type T_1 or T_2 respectively.

We can express this in our informal proof theoretic notation as (8).

(8) For Γ a system of complex types with strings, length determining string types and concatenative string types

a. $$\frac{\Gamma \vdash T_1 \in \textbf{Type} \qquad \Gamma \vdash T_2 \in \textbf{Type}}{\Gamma \vdash T_1{}^\frown T_2 \in \textbf{Type}}$$

b. $$\frac{\Gamma \vdash s_1 : T_1' \qquad \Gamma \vdash s_2 : T_2'}{\Gamma \vdash s_1 s_2 : T_1^\frown T_2} \; T_i' = T_i \text{ if } T_i \text{ is a string type; otherwise } T_i' = T_i^{=1}$$

(8a) introduces concatenative types, $T_1^\frown T_2$, and (8b) tells us that witnesses for these types are concatenations of two strings, s_1 and s_2, where $s_1 : T_1$ (or $s_1 : T_1^{=1}$ if T_1 is not a string type) and similarly for s_2 and T_2.

To (8) we can add (9) to express the associativity of '$^\frown$'.

(9) $$\frac{\Gamma \vdash (T_1^\frown T_2)^\frown T_3 \in \textbf{Type} \qquad \Gamma \vdash T_1^\frown(T_2^\frown T_3) \in \textbf{Type}}{\Gamma \vdash (T_1^\frown T_2)^\frown T_3 = T_1^\frown(T_2^\frown T_3) \in \textbf{Type}}$$

While strings as we have defined them are useful for modelling events in terms of strings of subevents, it is not the case that all strings of events can be considered as occurring events. Consider a case where we have two events as characterized in (10).

(10) a. $e_1 : T_1$
b. $e_2 : T_2$

The type theory as we have defined it will yield both judgements in (11).

(11) a. $e_1e_2 : T_1^\frown T_2$
b. $e_2e_1 : T_2^\frown T_1$

However, if we are using event strings to model temporal ordering it cannot be the case that both e_1e_2 and e_2e_1 both model occurring events even though they are both strings of events. That is, if e_1 temporally precedes e_2 then e_2 cannot temporally precede e_1 and *vice versa*. This has to do with the fact that a particular event can only happen once. It is, of course, possible to have different events of the same types occurring in the reverse order as in (12).

(12) a. $e_1e_2 : T_1^\frown T_2$
b. $e_3e_4 : T_2^\frown T_1$

One way to distinguish those strings which correspond to occurring events from those which do not is to introduce a type *Occur* such that if e_1e_2 : *Occur* then e_2e_1 :/*Occur*. Below we will use records to model the simultaneous occurrence of events. We can do this by allowing records to be witnesses for *Occur* and requiring that if r : *Occur* and $\ell_1, \ell_2 \in \text{labels}(r)$ then $r.\ell_1 r.\ell_2$:/*Occur*. We will not develop this idea further here but assume it in the background.

Let us return to Kim watching the boy, a, playing fetch with the dog, b, using the stick, c. She perceives the event as being of type play_fetch(a,b,c). But what does it mean to be an event of this type? Given our concatenation types we can build a type which corresponds to most of what we have sketched in Figure 2.1, namely (13).

(13) pick_up(a,c)$^\frown$attract_attention(a,b)$^\frown$throw(a,c)$^\frown$run_after(b,c)$^\frown$pick_up(b,c)$^\frown$
return(b,c,a)

(13) is a type corresponding to everything we have represented in Figure 2.1 except for the arrow which loops back from the end state to the start state. In order to get the loop into the event type we will use a Kleene-+ type. The type (14) will, then, give us a type corresponding to the complete Figure 2.1 since it will be the type consisting of strings of one or more events of the type (13).

(14) (pick_up(a,c)$^\frown$attract_attention(a,b)$^\frown$throw(a,c)$^\frown$run_after(b,c)$^\frown$pick_up(b,c)$^\frown$ return(b,c,a))$^+$

We will complicate (14) slightly by substituting record types for the ptypes as in (15). We do this because we will want to allow for things happening simultaneously and record types will give us a straightforward way of allowing this.

(15) ($\big[$e:pick_up(a,c)$\big]^\frown\big[$e:attract_attention(a,b)$\big]^\frown\big[$e:throw(a,c)$\big]^\frown\big[$e:run_after(b,c)$\big]^\frown$ $\big[$e:pick_up(b,c)$\big]^\frown\big[$e:return(b,c,a)$\big]$)$^+$

The label ‘e’ (“event”) occurs in each of the elements of the string type. In this case we will say that ‘e’ labels a *dimension* of events of this type. The ‘e’-dimension can be thought of as the dimension which characterizes what is happening at each stage of the event.

2.3 Doing things with types

2.3.1 Type acts

The boy and the dog have to coordinate and interact in order to create an event of the game of fetch. This involves doing more with types than just making judgements. For example, when the dog observes the situation in which the boy raises the stick, it may not be clear to the dog whether this is part of a fetch-game situation or a stick-beating situation. The dog may be in a situation of entertaining these two types as possibilities prior to making the judgement that the situation is of the fetch type. We will call this act a query as opposed to a judgement. Once the dog has made the judgement that what it has observed so far is an initial segment of a fetch type situation it has to make its own contribution in order to realize the fetch type, that is, it has to run after the stick and bring it back. This involves the creation of a situation of a certain type. Thus creation acts are another kind of act related to types. Creating objects of a given type often has a *de se* (see, for example, Perry, 1979; Lewis, 1979a; Ninan, 2010; Schlenker, 2011) aspect. The dog has to know that it itself must run after the stick in order to make this a situation in which it and the boy are playing fetch. There is something akin to what Perry calls an essential indexical here, though, of course, the dog does not have indexical linguistic expressions. It is nevertheless part of the basic competence that an agent needs in order to be able to coordinate its action with the rest of the world that it has a primitive sense of self which is distinct from being able to identify an object which has the same properties as itself. We will follow Lewis in modelling *de se* in terms of functional abstraction over the “self”. In our terms this will mean that *de se* type acts involve dependent types.

In standard type theory we have judgements such as $o : T$ “o is of type T” and T *true* “there is something of type T”. We want to enhance this notion of judgement by including a reference to the agent A which makes the judgement, giving judgements such as $o :_A T$ "agent A judges that o is of type T" and $:_A T$ "agent A judges that there is some object of type T". We will call the first of these a *specific* judgement and the second a *non-specific* judgement. Such judgements are one of the three kinds of acts represented in (16) that we want to include in our type act theory.

(16) *Type Acts*

judgements

specific $o :_A T$ "agent A judges object o to be of type T"

non-specific $:_A T$ "agent A judges that there is some object of type T"

queries

specific $o :_A T?$ "agent A wonders whether object o is of type T"

non-specific $:_A T?$ "agent A wonders whether there is some object of type T"

creations

non-specific $:_A T!$ "agent A creates something of type T"

Note that creations only come in the non-specific variant. You cannot create an object which already exists.

Creations are also limited in that there are certain types which a given agent is not able to realize as the main actor. Consider for example the event type involved in the fetch game of the dog running after the stick. The human cannot be the main creator of such an event since it is the dog who is the actor. The most the human can do is wait until the dog has carried out the action and we will count this as a creation type act. This will become important when we discuss coordination in the fetch-game below. It is actually important that the human makes this passive contribution to the creation of the event of the dog running after the stick and does not, for example, get the game confused by immediately throwing another stick before the dog has had a chance to retrieve the first stick. There are other cases of event types which require a less passive contribution from an agent other than the main actor. Consider the type of event where the dog returns the stick to the human. The dog is clearly the main actor here but the human has also a role to play in making the event realized. For example, if the human turns her back on the dog and ignores what is happening or runs away, the event type will not be realized despite the dog's best efforts. Other event types, such as lifting a piano, involve more equal collaboration between two or more agents, where it is not intuitively clear that any one of the agents is the main actor. So when we say "agent A creates something of type T" perhaps it would be more accurate to phrase this as "agent A contributes to the creation of something of type T" where A's contribution might be as little as not realizing any of the other types involved in the game until T has been realized.

De se type acts involve functions which have the agent in its domain and return a type, that is, they are dependent types which, given the agent, will yield a type. We will say that agents are of type *Ind* and that the relevant dependent types, $\mathcal{T}$, are functions of type (*Ind*→*Type*). We characterize *de se* type acts in a way parallel to (16), as given in (17).

(17) *De Se Type Acts*

judgements

specific $o :_A \mathcal{T}(A)$ "agent A judges object o to be of type $\mathcal{T}(A)$"

non-specific $:_A \mathcal{T}(A)$ "agent A judges that there is some object of type $\mathcal{T}(A)$"

queries

specific $o :_A \mathcal{T}(A)?$ "agent A wonders whether object o is of type $\mathcal{T}(A)$"

non-specific $:_A \mathcal{T}(A)?$ "agent A wonders whether there is some object of type $\mathcal{T}(A)$"

creations

non-specific $:_A \mathcal{T}(A)!$ "agent A creates something of type $\mathcal{T}(A)$"

From the point of view of the type theory *de se* type acts seem more complex than non-*de se* type acts since they involve a dependent rather than a non-dependent type and a functional application of that dependent type to the agent. However, from a cognitive perspective one might expect *de se* type acts to be more basic. Agents which perform type acts using types directly related to themselves are behaving egocentrically and one could regard it as a more advanced level of abstraction to consider types which are independent of the agent. This seems a puzzling way in which our notions of type seem in conflict with out intuitions about cognition.

While these type acts are prelinguistic (we need them to account for the dog's behaviour in the game of fetch), it seems that they are the basis on which the notion of speech act (Austin, 1962; Searle, 1969, and much subsequent literature) is built.

2.3.2 Making inferences about events

What happens when Kim perceives an event as being of the type (15)? She makes a series of observations of events, assigning them to types in the string type. Note that the ptypes in each of the types can be further broken down in a similar way. This gives us a whole hierarchy of perceived events which at some point have to bottom out in basic perceptions which are not further analyzed. In order to recognize an event as being of this type Kim does not need to perceive a string of events corresponding to each of the types in the string types. She may, for example, observe the boy waving the stick to attract the dog's attention, get distracted by a bird flying overhead for a while, and then return to the fetch event at the point where the dog is running back to the boy with the stick. This still enables her to perceive the event as an event of fetch playing because she has seen such events before and learned that such events are of the string type in (15). It suffices for her to observe enough of the elements in the string to distinguish the event from other event types she may have available in her knowledge resources. Suppose, for example, that she has just two event string types available that begin with the picking up of a stick by a human in the company of a dog. One is (15). The other is one that leads to the human beating the dog with the stick. If she only observes the picking up of the stick she cannot be sure whether what she is observing is a game of fetch or a beating. However, as soon as she observes something in the event string which belongs only to the fetch type she can reasonably conclude that she is observing an event of the fetch type. She may, of course, be wrong. She may be observing an event of a type which she does not yet have available in her resource of event types, in which case she will need to learn about the new event type and add it to her resources. However, given the resources at her disposal she can make a prediction about the nature of the rest of the event. One could model her prediction making ability in terms of a function which maps a situation (modelled as a record) to a type of predicted situation, for example (18).

(18) $$\lambda r{:}\begin{bmatrix}\text{x}{:}\textit{Ind}\\ \text{c}_{\text{human}}{:}\text{human(x)}\\ \text{y}{:}\textit{Ind}\\ \text{c}_{\text{dog}}{:}\text{dog(y)}\\ \text{z}{:}\textit{Ind}\\ \text{c}_{\text{stick}}{:}\text{stick(z)}\\ \text{e}{:}\begin{bmatrix}\text{e:pick\_up(x,z)}\end{bmatrix}{}^\frown\begin{bmatrix}\text{e:attract\_attention(x,y)}\end{bmatrix}\end{bmatrix}\ .\ \begin{bmatrix}\text{e:play\_fetch}(r\text{.x},r\text{.y},r\text{.z})\\ \text{c}_{\text{init}}{:}\text{init}(r\text{.e,e})\end{bmatrix}$$

Here the predicate 'init' has arity ⟨*String*, *String*⟩. The type init(s_1,s_2) is non-empty just in case s_1 is an initial substring of s_2.

We achieve this by the definition in (19), repeated in Appendix 19.

(19) If s_1 is a string of length n and s_2 is a string of any length, then s : init(s_1,s_2) iff the length of s_2 is greater than or equal to n and for each i, $0 \leq i < n$, $s_1[i] = s_2[i]$ and $s = s_2$.

That is, if the initial substring condition holds then the second argument to the predicate (and nothing else) is of the ptype.

The kind of function of which (18 is an instance is a function of the general form (20).

(20) $\lambda a{:}T_1\ .\ T_2(\!(a)\!)$

Recall that the notation $T_2(\!(a)\!)$ represents that T_2 depends on a. The nature of this dependence in (18) is seen in the occurrences of r in the body of the function, for example, (21).

(21) play_fetch(r.x,r.y,r.z)

A function of the form (20) maps an object of some type (represented by T_1) to a type (represented by $T_2(\!(a)\!)$). The type that results from an application of this function will depend on what object it is applied to – that is, we have the possibility of obtaining different types from different objects. This function is then a dependent type as discussed, for example, on p. 27, These functions will play an important role in much of what is to come later in this book. They will show up many times in what appear at first blush to be totally unrelated phenomena. We want to suggest, however, that all of the phenomena we will describe using such functions have their origin in our basic cognitive ability to make predictions on the basis of partial observation of objects and events.

Functions which are dependent types return types but they do not, of themselves, tell us what to do with the type if we have obtained it by applying the function to an argument. Suppose $\mathcal{T}$ is the function (18) and that T is the domain type of $\mathcal{T}$, that is, (22).

$$(22)\quad \begin{bmatrix} \text{x}:\textit{Ind} \\ \text{c}_{\text{human}}:\text{human(x)} \\ \text{y}:\textit{Ind} \\ \text{c}_{\text{dog}}:\text{dog(y)} \\ \text{z}:\textit{Ind} \\ \text{c}_{\text{stick}}:\text{stick(z)} \\ \text{e}:\begin{bmatrix}\text{e:pick\_up(x,z)}\end{bmatrix}^\frown\begin{bmatrix}\text{e:attract\_attention(x,y)}\end{bmatrix} \end{bmatrix}$$

Then we may have the action rule given in (23).

$$(23)\quad \frac{s :_A T}{:_A \mathcal{T}(s)}$$

(23) represents that if an agent, A, judges a situation, s, to be of type T then A is licensed to judge that there is some situation of type $\mathcal{T}(s)$. We use a wavy line in this inference rule to indicate that it does not represent a conclusion that follows from a premise in a logical sense, but rather that the act above the line licenses the act below the line. That is, on the basis of what is above the line it is reasonable to perform what is below the line, though without a guarantee that it is correct or even that the action will be performed. Given that you observe a human pick up a stick and attract a dog's attention, it is reasonable to conclude that there will be an event of playing

fetch, but there is no guarantee that there actually will be such an event. We have talked in terms of what is above the line licensing what is below the line. Another term that can be used is *afford* which goes back to Gibson's (1979) notion of affordance. Thus we can talk of the action above the line as affording the action below the line.

Sometimes dependent type functions like $\mathcal{T}$ can be associated with more than one action rule. Agents may get to choose which they apply or perhaps there will be aspects of the context which will determine which of the action rules is appropriate. Thus in addition to (23) we might also have the action rules in (24).

(24) a. $$\frac{s :_A T}{s :_A \mathcal{T}(s)}$$

b. $$\frac{s :_A T}{:_A \mathcal{T}(s)!}$$

(24a) says that if A judges s to be of type T then A is licensed to judge that s is also of type $\mathcal{T}(s)$. (24b) says that if A judges s to be of type T then A is licensed to create something of type $\mathcal{T}(s)$.

What happens when Kim does not observe enough of the event to be able to predict with any certainty that the complete event will be a game of fetch? One theory would be that she can only make categorical judgements, and that she has to wait until she has seen enough so that there is only one type that matches in the collection of situation types in her resources. Another theory would be one where she predicts a disjunction of the available matching types when there is more than one that matches. One might refine this theory so that she can choose one of the available types but assign it a probability based on the number of matching types. If n is the number of matching types the probability of any one of them might be $\frac{1}{n}$. This assumes that each of the types is equally likely to be realized. It would be natural to assume, however, that the probability which Kim assigns to any one of the matching types would be dependent on her previous experience. Suppose, for example, that she has seen 100 events of a boy picking up a stick in the company of a dog, 99 of those events led to a game of fetch and only one led to the boy beating the dog. One might then assume that when she now sees the boy pick up the stick she would assign a .99 probability (on a scale of 0 to 1) to the type of fetch events and only .01 probability to the boy beating the dog. That is, the probability she assigns to an event of a boy picking up a stick leading to a game of fetch is the result of dividing the number of instances of a game of fetch she has already observed by the sum of the number of instances she has observed of any types whose initial segment involves the picking up of a stick. In more general terms we can compute the probability which an agent A assigns on the basis of a string, s, of previous observations to a predicted type T_{pr} given an observed type T_{obs}, $p_{A,\omega}(T_{pr} \mid T_{obs})$, in the case where T_{pr} is a member of the set of alternatives which can be predicted from T_{obs} according to A's resources based on p, $\text{alt}_{A,s}(T_{obs})$, by the formula in (25).

(25) $p_{A,s}(T_{pr} \mid T_{obs}) = \frac{|\{T_{pr}\}^{A,s}|}{\sum\limits_{T_{alt} \in \mathrm{alt}_{A,s}(T_{obs})} |\{T_{alt}\}^{A,s}|}$

where $\{T\}^{A,s}$ is the set of objects of type T observed by A in s. If T_{pr} is not a member of $\mathrm{alt}_{A,s}(T_{obs})$, that is not one of the alternatives, we say that $p_{A,s}(T_{pr} \mid T_{obs}) = 0$.

Where does the set of alternatives come from? We assume that an agent has a set of functions similar to (18) available as cognitive resources, that is, a set of resources that associates objects of given types with another type, that is collections of dependent types. We could think of these resources as topoi in the sense of Breitholtz (fthc). Among this collection of functions may be several which share the same domain type, that is, for some particular type T they are witnesses of the function type $(T \rightarrow \mathit{Type})$. Suppose that F is a set of such resources sharing the type T as a domain type. Then the set of alternatives for an object r of type T with respect to F is $\{T' \mid T' = f(r) \text{ for some } f \in F\}$.

While this is still a rather naive and simple view of how probabilities might be assigned it is not without interest, as shown by the following points:

Probability distributions It will always provide a probability distribution over sets of alternatives, that is (26).

(26) $\sum\limits_{T_{pr} \in \mathrm{alt}_{A,s}(T_{obs})} p_{A,s}(T_{pr} \mid T_{obs}) = 1$

Alternatives We have assumed a notion of alternatives based on types of completed events for which the observed event is an initial segment but other notions of alternativeness could be considered and perhaps even combined.

Relativity of probability assignments The notion of probability is both agent and resource relative. It represents the probability which an agent will assign to a type when observing a given situation after a previous string of observations. Two agents may assign different probabilities depending on the resources they have available.

Learning Relevant observations will update the probability distributions an agent will assign to a given set of alternatives since the probability is computed on the basis of previous observations of the alternative types.

Kim is not alone in being able to draw conclusions based on partial observations of an event. The dog can do it too. As soon as the boy has raised the stick and attracted the dog's attention the dog is excitedly snapping at the stick and starting to run in the direction in which the boy seems to be about to throw. The dog also seems to be attuned to string types of events just as Kim is and also able to make predictions on the basis of partial observations. The types to which a dog is attuned will not be the same as those to which humans can be attuned and this can certainly lead to miscommunication between humans and dogs. For example, there may be many reasons why I would go to the place where outdoor clothes are hanging and where the dog's lead is kept. Many times it will be because I am planning to take the dog out for a walk, but not as often as the dog appears to think, judging from the excitement he shows any time I go near the lead. It is difficult to explain to the dog that I am just looking for a receipt that I think I might have left in my coat pocket. But the basic mechanism of being able to assemble types of events into string types of more complex events and make predictions on the basis of these types seems to be common to both humans and dogs and a good number of other animals too. Perhaps simple organisms do not have this ability and can only react to events that have already happened, but not to predicted outcomes.

This basic inferential ability is thus not parasitic on the ability to communicate using a human language. It is, however, an ability which appears to be exploited to a great extent in our use of language as we will see in later chapters.

2.3.3 Coordination and games

Let us now apply these notions to the kind of interaction that has to take place between the human and the dog in a game of fetch. First consider in more detail what is actually involved in playing a game of fetch, that is creating an event of type (15). Each agent has to keep track in some way of where they are in the game and in particular what needs to happen next. We analyze this by saying that each agent has an information state which we will model as a record. We need to keep track of the progression of types of information state for an agent during the course of the game. We will refer to the types of information states as gameboards.[2] The idea is that as part of the event occurs, the agent's gameboard is updated so that an event of the next type in the string is expected. For now, we will consider gameboards which only place one requirement on information states, namely that there is an agenda which indicates the type of the next move in the game. Thus if the agent is playing fetch and observes an event of the type where the human throws the stick, then, according to (15), the next move in the game will be an event of the type where the dog runs after the stick. If the actor in the next move is the agent herself then the agent will need to create an event of the type of the next move if the game is to progress. If the actor in the next move is the other player in the game, then the agent will need to observe an event and judge it to be of the appropriate type in order for the game to progress. The type of information states, *InfoState*, will be (27a). The type of the initial information state, *InitInfoState*, will be one where the agenda is required to be the empty list.

(27) a. $\left[\ \text{agenda} \ : \ \text{list}(\mathit{RecType})\ \right]$

[2]Our notions of *information state* and *gameboard* are taken from Larsson (2002) and Ginzburg (2012) respectively as well as a great deal of related literature on the gameboard or information state approach to dialogue analysis originating from Ginzburg (1994). We have adapted the notions somewhat to our own purposes.

b. $\left[\ \text{agenda=[ ]} \ : \ \text{list}(\textit{RecType})\ \right]$

The type *RecType* is the type of record types, that is, the witnesses for this type will be record types. Just like the type *Type*, *RecType* should have a superscript as in *RecType*^{n}, representing the order with which it is associated in the stratification of the type system. This was made precise in Chapter 1, p. 37ff. As with *Type*, we ignore the stratification order superscript except where it becomes important to mention it. For any type, T, $\text{list}(T)$ is also a type, the type whose witnesses are lists all of whose members are of type T. Suppose that a, b, c are of type T, then the list $[a, b, c]$ is of type $\text{list}(T)$. We use [ ], as in (27b) to represent the empty list, identical with the empty set, $\emptyset$. The field in this type is an example of a manifest field (Coquand *et al.*, 2004). We use the notation $\left[\ell{=}a{:}T\right]$ to represent $\left[\ell{:}T_a\right]$ where T_a is a type such that $b : T_a$ just in case $b : T$ and $b = a$, that is it either has no witnesses because a is not of type T or it has exactly one witness, a, of type T. We call T_a a *singleton type*.

$[a, b, c]$ is a convenient standard notation for the list consisting of a, b and c in that order but, as with strings, we will actually model lists as records with a first and rest structure as in (28) and we will use the standard notation as a convenient abbreviation.

(28) $$\begin{bmatrix} \text{fst} & = & a \\ \text{rst} & = & \begin{bmatrix} \text{fst} & = & b \\ \text{rst} & = & \begin{bmatrix} \text{fst} & = & c \\ \text{rst} & = & [\,] \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

We introduce list types and singleton types in detail below.

We can introduce lists and list types into our type systems by the definition in (29), repeated in Appendix 18.

(29) A system of complex types with record types $\textbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has list types* if

1. for any $T \in$ **Type**, $\text{list}(T) \in$ **Type**
2. for any $T \in$ **Type**,
   a) $[\,] :_{\mathbf{TYPE_C}} \text{list}(T)$
   b) $a\,|\,L :_{\mathbf{TYPE_C}} \text{list}(T)$ iff $a :_{\mathbf{TYPE_C}} T$ and $L :_{\mathbf{TYPE_C}} \text{list}(T)$

Lists are a common data structure used in computer science but they are not normally defined in basic set theory, although it is straightforward to define them in terms of records. We will use the reserved labels ‘fst’ and ‘rst’ for the first member of the list and the remainder (“rest”) of the list respectively. We let the empty list, [], be the empty set, $\emptyset$. If L is a list then $a\,|\,L$ is to be the record in (30).

(30) $\left[\begin{array}{lcl} \text{fst} & = & a \\ \text{rst} & = & L \end{array}\right]$

If L is a list we often use $\text{fst}(L)$ and $\text{rst}(L)$ to represent L.fst and L.rst respectively.

In our informal proof theoretic notation we can characterize type systems with lists as in (31).

(31) For Γ a system of complex types with list types

a. $\dfrac{\Gamma \vdash T \in \textbf{Type}}{\Gamma \vdash \text{list}(T) \in \textbf{Type}}$

b. $\dfrac{}{\Gamma \vdash [\,] : \text{list}(T)}$ $\qquad$ $\dfrac{\Gamma \vdash a : T \qquad \Gamma \vdash L : \text{list}(T)}{\Gamma \vdash a\,|\,L : \text{list}(T)}$

(31a) introduces list types and corresponds to clause 1 of (29). (31b) gives an inductive definitions of the set of witnesses for an arbitrary list type and corresponds to clause 2 of (29).

We introduce singleton types by the definition in (32), repeated in Appendix 6.

(32) A system of complex types $\textbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has singleton types* if

1. for any $T, T' \in$ **Type** and $a :_{\textbf{TYPE}_\textbf{C}} T'$, $T_a \in$ **Type**
2. for any $T, T' \in$ **Type** and $a :_{\textbf{TYPE}_\textbf{C}} T'$, $b :_{\textbf{TYPE}_\textbf{C}} T_a$ iff $b :_{\textbf{TYPE}_\textbf{C}} T$ and $a = b$

Note that this definition allows the formation of singleton types from singleton types. We sometimes refer to these as *multiple* singleton types and notate them as $T_{a,b,\ldots}$. Following the definition above an object c will be of type $T_{a,b}$ just in case $c : T$ and $a = b = c$.

In our informal proof theoretic notation we can characterize type systems with singleton types as in (33).

(33) For Γ a system of complex types with singleton types

a. $$\frac{\Gamma \vdash T \in \textbf{Type} \qquad \Gamma \vdash a : T'}{\Gamma \vdash T_a \in \textbf{Type}}$$

b. $$\frac{\Gamma \vdash a = b : T}{\Gamma \vdash b : T_a}$$

We can now see the rules of the game corresponding to the type (15) as a set of update functions which indicate for an information state of a given type what type the next information state may belong to if an event of a certain type occurs. These update functions correspond to the transitions in a finite state machine. This is given in (34).

(34)
$$\begin{array}{l}
\{\ \lambda r{:}\big[\text{agenda=[]:list}(\textit{RecType})\big]\ .\\
\quad\quad \big[\text{agenda=[}\big[\text{e:pick_up}(a,c)\big]\text{]:list}(\textit{RecType})\big],\\
\quad \lambda r{:}\big[\text{agenda=[}\big[\text{e:pick_up}(a,c)\big]\text{]:list}(\textit{RecType})\big]\\
\quad\quad \lambda e{:}\big[\text{e:pick_up}(a,c)\big]\ .\\
\quad\quad\quad \big[\text{agenda=[}\big[\text{e:attract_attention}(a,b)\big]\text{]:list}(\textit{RecType})\big],\\
\quad \lambda r{:}\big[\text{agenda=[}\big[\text{e:attract_attention}(a,b)\big]\text{]:list}(\textit{RecType})\big]\\
\quad\quad \lambda e{:}\big[\text{e:attract_attention}(a,b)\big]\ .\\
\quad\quad\quad \big[\text{agenda=[}\big[\text{e:throw}(a,c)\big]\text{]:list}(\textit{RecType})\big],\\
\quad \lambda r{:}\big[\text{agenda=[}\big[\text{e:throw}(a,c)\big]\text{]:list}(\textit{RecType})\big]\\
\quad\quad \lambda e{:}\big[\text{e:throw}(a,c)\big]\ .\\
\quad\quad\quad \big[\text{agenda=[}\big[\text{e:run_after}(b,c)\big]\text{]:list}(\textit{RecType})\big],\\
\quad \lambda r{:}\big[\text{agenda=[}\big[\text{e:run_after}(b,c)\big]\text{]:list}(\textit{RecType})\big]\\
\quad\quad \lambda e{:}\big[\text{e:run_after}(b,c)\big]\ .\\
\quad\quad\quad \big[\text{agenda=[}\big[\text{e:pick_up}(b,c)\big]\text{]:list}(\textit{RecType})\big],\\
\quad \lambda r{:}\big[\text{agenda=[}\big[\text{e:pick_up}(b,c)\big]\text{]:list}(\textit{RecType})\big]\\
\quad\quad \lambda e{:}\big[\text{e:pick_up}(b,c)\big]\ .\\
\quad\quad\quad \big[\text{agenda=[}\big[\text{e:return}(b,c,a)\big]\text{]:list}(\textit{RecType})\big],\\
\quad \lambda r{:}\big[\text{agenda=[}\big[\text{e:return}(b,c,a)\big]\text{]:list}(\textit{RecType})\big]\\
\quad\quad \lambda e{:}\big[\text{e:return}(b,c,a)\big]\ .\\
\quad\quad\quad \big[\text{agenda=[]:list}(\textit{RecType})\big]\\
\hspace{20em}\}
\end{array}$$

Since we are treating an empty agenda as the condition for the input to the initial state in the corresponding automaton and also the output of the final state we automatically get the loop effect from the final state to the initial state. In order to prevent the loop we would have to distinguish the type corresponding to the initial and final states.

The first function listed in (34) is of the type (35).

(35) (*InitInfoState*→*RecType*)

It maps an initial information state, that is, with an empty agenda, to a record type where the type of event where the human, a, picks up the stick c is on the agenda.

The remaining functions all map information states of some type to a function. For example, the first of these functions is of the type in (36).

(36) $((r{:}\left[\text{agenda=[}\left[\text{e:pick\_up}(a,c)\right]\text{]:list}(\textit{RecType})\right])\rightarrow(\text{fst}(r.\text{agenda})\rightarrow\textit{RecType}))$

They map an information state where some type is on the agenda and an event of that type to a new type of information state where the next type to be realized in the game is on the agenda. (36) is a dependent function type. A function of this type maps something, r, of type $\left[\text{agenda=[}\left[\text{e:pick\_up}(a,c)\right]\text{]:list}(\textit{RecType})\right]$, to a function from records of the first type on the list in the 'agenda'-field in r to a record type. For any list, L, we use $\text{fst}(L)$ to represent the first member of L. Dependent functions are slightly more complex than the function types we have seen previously in that they introduce a variable (in this case, r) on which type of the object they return can depend. Schematically, we can represent the difference between non-dependent function types and dependent function types as the difference between $(T_1 \rightarrow T_2)$ and $(a : T_1) \rightarrow T_2(\!(a)\!)$.

We can introduce dependent function types into our type systems as in (37), repeated in Appendix 10.

(37) An intensional system of complex types $\mathbf{TYPE}_{IC}$,

$$\mathbf{TYPE}_{IC} = \langle \mathbf{Type}^n, \mathbf{BType}, \langle \mathbf{PType}^n, \mathbf{Pred}, \mathbf{ArgIndices}, \textit{Arity}\rangle, \langle A, F^n\rangle\rangle_{n\in \textit{Nat}}$$

has dependent function types if

1. for any $n > 0$, $T \in \mathbf{Type}^n$ and $\mathcal{T} :_{\mathbf{TYPE}_{IC_n}} (T \rightarrow \textit{Type}^n)$, $((a : T) \rightarrow \mathcal{T}(a)) \in \mathbf{Type}^n$
2. for each $n > 0$, $f :_{\mathbf{TYPE}_{IC_n}} ((a : T) \rightarrow \mathcal{T}(a))$ iff f is a function whose domain is $\{a \mid a :_{\mathbf{TYPE}_{IC_n}} T\}$ and such that for any a in the domain of f, $f(a) :_{\mathbf{TYPE}_{IC_n}} \mathcal{T}(a)$.

We can express this in our formal proof theoretic notation as (38).

(38) For $\{\Gamma^n\}_{n\in \textit{Nat}}$ an intensional system with complex types and dependent function types

a. $$\frac{\Gamma^n \vdash T \in \textbf{Type}^n \qquad \Gamma^n \vdash \mathcal{T} : (T \rightarrow \mathit{Type}^n)}{\Gamma^n \vdash ((a : T) \rightarrow \mathcal{T}(a)) \in \textbf{Type}^n}$$

b. $$\frac{\begin{array}{c}[\Gamma^n \vdash a : T] \\ \vdots \\ \Gamma^n \vdash f(a) : \mathcal{T}(a)\end{array}}{\Gamma^n \vdash f : ((a : T) \rightarrow \mathcal{T}(a))} \qquad \frac{\Gamma^n \vdash f : ((a : T) \rightarrow \mathcal{T}(a)) \qquad \Gamma^n \vdash f(a) : \mathcal{T}(a)}{\Gamma^n \vdash a : T}$$

(38a) introduces dependent function types. (38b) characterizes what it means for a function to be a witness for a dependent function type. The first rule says that if, assuming $a : T$, we can conclude that $f(a) : \mathcal{T}(a)$, then f is of the dependent function type $((a : T) \rightarrow \mathcal{T}(a))$. The second rule says that if a function f is of this type and f applied to some object a is of type $\mathcal{T}(a)$ then $a : T$. We have included this second rule in order to require that the domain type of f is T. The first rule requires that f will be defined on anything of type T and the second requires that anything the function is defined on is of type T, that is, a function f of type $((a : T) \rightarrow \mathcal{T}(a))$ is defined on all and only witnesses of T.

We can think of the set (34) of update functions as the set of rules which define the game. With the types we have so far these rules will not have a type in common. All the rules, except for the first one listed will be defined on some type of information state with a non-empty agenda. If T is a type we will use $\mathrm{nelist}(T)$ to represent the type of non-empty lists of objects of type T given in (39).

(39) $$\begin{bmatrix} \text{fst} & : & T \\ \text{rst} & : & \mathrm{list}(T) \end{bmatrix}$$

We might think that all the functions listed in (34) are of the type (40).

(40) $((r{:}\big[\text{agenda}{:}\mathrm{nelist}(\mathit{RecType})\big]))\rightarrow(\mathrm{fst}(r.\text{agenda})\rightarrow\mathit{RecType}))$

This would, however, be incorrect since functions of the type (40) would have to be defined on all information states with a non-empty agenda. However, the functions in (34) are only defined on some of the information states with a non-empty agenda, since they require a particular type to be on the agenda. That is they are *partial* functions on information states with non-empty agenda. We shall use $(T_1 \rightharpoonup T_2)$ to represent the type of partial functions from objects of type T_1 to objects of type T_2. Functions of this type are not arbitrary partial functions but those which are of some total function type $(T \rightarrow T_2)$ such that any object of type T is also of type T_1. That is, we only consider partial functions which are total functions on the objects of some type.

We can characterize this formally by (41), repeated in Appendix 4.

(41) A system of complex types $\textbf{TYPE}_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ with function types *has partial function types* if

1. for any $T_1, T_2 \in$ **Type**, $(T_1 \rightharpoonup T_2) \in$ **Type**
2. for any $T_1, T_2 \in$ **Type**, $f :_{\mathbf{TYPE_C}} (T_1 \rightharpoonup T_2)$ iff there is some type T' such that $f : (T' \rightarrow T_2)$ and for any a, if $a : T'$ then $a : T_1$

This can be expressed in our informal proof theoretic notation as (42).

(42) For Γ a system of complex types with partial function types:

a. $$\frac{\Gamma \vdash T_1 \in \textbf{Type} \qquad \Gamma \vdash T_2 \in \textbf{Type}}{\Gamma \vdash (T_1 \rightharpoonup T_2) \in \textbf{Type}}$$

b. $$\frac{\Gamma \vdash f : (T \rightarrow T_2) \qquad \begin{array}{c}[\Gamma \vdash x : T] \\ \vdots \\ \Gamma \vdash x : T_1\end{array}}{\Gamma \vdash f : (T_1 \rightharpoonup T_2)}$$

$$\frac{\Gamma \vdash f : (T_1 \rightharpoonup T_2) \qquad a \in \mathrm{dom}(f)}{\Gamma \vdash a : T_1} \qquad \frac{\Gamma \vdash f : (T_1 \rightharpoonup T_2) \qquad a \in \mathrm{dom}(f)}{\Gamma \vdash f(a) : T_2}$$

Essentially these definitions characterize a type $(T_1 \rightharpoonup T_2)$ as a limited kind of *polymorphic* function type. That is, a function will be of this type just in case it is a witness for one of many types $(T \rightarrow T_2)$ where any witness for type T is also a witness for type T_1.

The rules in the game (34) are either of type (35) or (43).

(43) $((r{:}\left[\text{agenda:}\mathrm{nelist}(\textit{RecType})\right]))\rightharpoonup(\mathrm{fst}(r.\text{agenda})\rightarrow\textit{RecType}))$

That is, they are all of the *join type* in (44), which we will call *GameRule*.

(44) $((\textit{InitInfoState}\rightarrow\textit{RecType})\vee((r{:}\left[\text{agenda:}\mathrm{nelist}(\textit{RecType})\right]))\rightharpoonup(\mathrm{fst}(r.\text{agenda})\rightarrow\textit{RecType})))$

In general we can say that for any two types, T_1 and T_2, there is another type $(T_1 \vee T_2)$ and that $a : (T_1 \vee T_2)$ just in case either $a : T_1$ or $a : T_2$. Join types can also be called union or disjunctive types.

We can introduce join types formally as in (45), repeated in Appendix 7.

(45) A system of complex types **TYPE**$_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has join types* if

1. for any $T_1, T_2 \in \mathbf{Type}$, $(T_1 \vee T_2) \in \mathbf{Type}$
2. for any $T_1, T_2 \in \mathbf{Type}$, $a :_{\mathbf{TYPE_C}} (T_1 \vee T_2)$ iff $a :_{\mathbf{TYPE_C}} T_1$ or $a :_{\mathbf{TYPE_C}} T_2$

In our informal proof theoretic notation this can be expressed as (46).

(46) For Γ a system of complex types with join types:

a. $$\frac{\Gamma \vdash T_1 \in \textbf{Type} \qquad \Gamma \vdash T_2 \in \textbf{Type}}{\Gamma \vdash (T_1 \vee T_2) \in \textbf{Type}}$$

b. $$\frac{\Gamma \vdash a : T_1 \qquad \Gamma \vdash T_2 \in \textbf{Type}}{\Gamma \vdash a : (T_1 \vee T_2)} \qquad \frac{\Gamma \vdash T_1 \in \textbf{Type} \qquad \Gamma \vdash a : T_2}{\Gamma \vdash a : (T_1 \vee T_2)}$$

$$\frac{\Gamma \vdash a : (T_1 \vee T_2) \qquad \begin{array}{c}[\Gamma \vdash x : T_1]\\ \vdots \\ \Gamma \vdash x : T\end{array} \qquad \begin{array}{c}[\Gamma \vdash x : T_2]\\ \vdots \\ \Gamma \vdash x : T\end{array}}{\Gamma \vdash a : T}$$

(46a) tells us that for any two types in the system there is a join type which can be constructed from them. The first two rules in (46b) tell us it is sufficient for an object to be a witness of one of the two types in order to be a witness for the join type. The third rule tells us that if something, a, is a witness for a join type and there is some type, T, such that we can show that witnesses for either of the two types used to construct the join type are also witnesses for T, then a is a witness for T.

We specify that $(T_1 \vee T_2)$ represents the labelled set (47)

(47) $\{\langle \mathrm{disj}_1, T_1\rangle, \langle \mathrm{disj}_2, T_2\rangle\}$

where ‘disj$_1$’ and ‘disj$_2$’ are reserved labels (“disjunct”). For many purposes it may be an unwanted consequence of this characterization of join types that the types $T_1 \vee T_2$ and $T_2 \vee T_1$

are distinct types, albeit with the same set of witnesses. For cases where this is not desired we introduce *generalized join types* as in (48), repeated in Appendix 7.

(48) A system of complex types **TYPE**$_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has generalized join types* if

1. for any finite set of types, $\mathfrak{T}$, such that $\mathfrak{T} \subseteq \mathbf{Type}$, $\bigvee \mathfrak{T} \in \mathbf{Type}$
2. for any finite $\mathfrak{T} \subseteq \mathbf{Type}$, $a :_{\mathbf{TYPE_C}} \bigvee \mathfrak{T}$ iff $a :_{\mathbf{TYPE_C}} T$ for some $T \in \mathfrak{T}$

In our informal proof theoretic notation, this is expressed as (49).

(49) For Γ a system of complex types with generalized join types:

a. $$\frac{\Gamma \vdash T_1 \in \textbf{Type}, T_2 \in \textbf{Type}, \ldots, T_n \in \textbf{Type}}{\Gamma \vdash \bigvee\{T_1, T_2, \ldots, T_n\} \in \textbf{Type}}$$

b. $$\frac{\Gamma \vdash \bigvee\{T_1, \ldots, T_i, \ldots, T_n\} \in \textbf{Type} \qquad \Gamma \vdash a : T_i}{\Gamma \vdash a : \bigvee\{T_1, \ldots, T_i, \ldots, T_n\}}$$

$$\frac{\Gamma \vdash a : \bigvee\{T_1, \ldots, T_n\} \qquad \begin{array}{c}[\Gamma \vdash x : T_1]\\ \vdots \\ \Gamma \vdash x : T\end{array} \quad \cdots \quad \begin{array}{c}[\Gamma \vdash x : T_n]\\ \vdots \\ \Gamma \vdash x : T\end{array}}{\Gamma \vdash a : T}$$

We can then, if desired, use the notation $(T_1 \vee T_2 \vee \ldots \vee T_n)$ to express $\bigvee\{T_1, T_2, \ldots, T_n\}$.

The set in (34) is thus a set all of whose members are witness of type *GameRule*, that is the set is of type $\mathrm{set}(\textit{GameRule})$. For any type, T, there is a type $\mathrm{set}(T)$ whose witnesses are sets each of whose members are of type T.

We can introduce set types formally as in (50), repeated in Appendix 5.

(50) A system of complex types **TYPE**$_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has set types* if

1. for any $T \in$ **Type**, $\mathrm{set}(T) \in$ **Type**
2. for any $T \in$ **Type**, $X :_{\mathbf{TYPE_C}} \mathrm{set}(T)$ iff X is a set and for all $a \in X$, $a :_{\mathbf{TYPE_C}} T$

In our informal proof theoretic notations, this is expressed as (51).

(51) For Γ a system of complex types with set types:

a. $$\frac{\Gamma \vdash T \in \textbf{Type}}{\Gamma \vdash \mathrm{set}(T) \in \textbf{Type}}$$

b. $$\frac{X \text{ set} \qquad \begin{matrix}[x \in X] \\ \vdots \\ \Gamma \vdash x : T\end{matrix}}{X : \mathrm{set}(T)} \qquad \frac{\Gamma \vdash X : \mathrm{set}(T) \qquad a \in X}{\Gamma \vdash a : T}$$

(51a) tells us that for any type, T, there is another type $\mathrm{set}(T)$. In (51b) we use 'X set' to represent that X is a set, in the sense of set theory. The first rule in (51b) tells that if we have a set, X, and a way of showing that any member of X is of type T, then X is of type $\mathrm{set}(T)$. The second rule tells us conversely that if we have a set of type $\mathrm{set}(T)$ then any member of X is of type T. The inference rules in (51b) are presumably not rules that would be available in a constructive type theory in that they rely on the set theoretic notion of set (including infinite and non-denumerable sets) and it may not be decidable whether all the members of an arbitrary set are of type T or not. One strategy that can be used, if we are concerned about this, is to limit the notion of set to those which are recursive. For many of the applications we have in mind, finite sets are adequate. For example, we would want probably want to limit games to having a finite set of rules.

The set of game rules in (34) gives the rules for specific participants, a, b and c. In order to characterize the game in general we need to abstract out the roles of the individual participants in the game. This we will do by defining a function from a record containing individuals appropriate to play the roles in the game thus revising (34) to (52).

(52) $$\lambda r^{*}:\begin{bmatrix} \text{h} & : & \textit{Ind} \\ \text{c}_{\text{human}} & : & \text{human(h)} \\ \text{d} & : & \textit{Ind} \\ \text{c}_{\text{dog}} & : & \text{dog(d)} \\ \text{s} & : & \textit{Ind} \\ \text{c}_{\text{stick}} & : & \text{stick(s)} \end{bmatrix} .$$

$$
\begin{aligned}
\{\ & \lambda r{:}\left[\text{agenda=[]:[}\textit{RecType}\text{]}\right] . \\
& \quad \left[\text{agenda=[}\left[\text{e:pick\_up}(r^*.\text{h},r^*.\text{s})\right]\text{]:[}\textit{RecType}\text{]}\right], \\
& \lambda r{:}\left[\text{agenda=[}\left[\text{e:pick\_up}(r^*.\text{h},r^*.\text{s})\right]\text{]:[}\textit{RecType}\text{]}\right] \\
& \quad \lambda e{:}\left[\text{e:pick\_up}(r^*.\text{h},r^*.\text{s})\right] . \\
& \qquad \left[\text{agenda=[}\left[\text{e:attract\_attention}(r^*.\text{h},r^*.\text{d})\right]\text{]:[}\textit{RecType}\text{]}\right], \\
& \lambda r{:}\left[\text{agenda=[}\left[\text{e:attract\_attention}(r^*.\text{h},r^*.\text{d})\right]\text{]:[}\textit{RecType}\text{]}\right] \\
& \quad \lambda e{:}\left[\text{e:attract\_attention}(r^*.\text{h},r^*.\text{d})\right] . \\
& \qquad \left[\text{agenda=[}\left[\text{e:throw}(r^*.\text{h},r^*.\text{s})\right]\text{]:[}\textit{RecType}\text{]}\right], \\
& \lambda r{:}\left[\text{agenda=[}\left[\text{e:throw}(r^*.\text{h},r^*.\text{s})\right]\text{]:[}\textit{RecType}\text{]}\right] \\
& \quad \lambda e{:}\left[\text{e:throw}(r^*.\text{h},r^*.\text{s})\right] . \\
& \qquad \left[\text{agenda=[}\left[\text{e:run\_after}(r^*.\text{d},r^*.\text{s})\right]\text{]:[}\textit{RecType}\text{]}\right], \\
& \lambda r{:}\left[\text{agenda=[}\left[\text{e:run\_after}(r^*.\text{d},r^*.\text{s})\right]\text{]:[}\textit{RecType}\text{]}\right] \\
& \quad \lambda e{:}\left[\text{e:run\_after}(r^*.\text{d},r^*.\text{s})\right] . \\
& \qquad \left[\text{agenda=[}\left[\text{e:pick\_up}(r^*.\text{d},r^*.\text{s})\right]\text{]:[}\textit{RecType}\text{]}\right], \\
& \lambda r{:}\left[\text{agenda=[}\left[\text{e:pick\_up}(r^*.\text{d},r^*.\text{s})\right]\text{]:[}\textit{RecType}\text{]}\right] \\
& \quad \lambda e{:}\left[\text{e:pick\_up}(r^*.\text{d},r^*.\text{s})\right] . \\
& \qquad \left[\text{agenda=[}\left[\text{e:return}(r^*.\text{d},r^*.\text{s},r^*.\text{h})\right]\text{]:[}\textit{RecType}\text{]}\right], \\
& \lambda r{:}\left[\text{agenda=[}\left[\text{e:return}(r^*.\text{d},r^*.\text{s},r^*.\text{h})\right]\text{]:[}\textit{RecType}\text{]}\right] \\
& \quad \lambda e{:}\left[\text{e:return}(r^*.\text{d},r^*.\text{s},r^*.\text{h})\right] . \\
& \qquad \left[\text{agenda=[]:[}\textit{RecType}\text{]}\right] \\
& \hspace{20em} \}
\end{aligned}
$$

(52) is of type $(\textit{Rec} \rightharpoonup \mathrm{set}(\textit{GameRule}))$ which we will call *Game*.

Specifying the rules of the game in terms of update functions in this way will not actually getting anything to happen, though. For that we need type acts of the kind we discussed. We link the update functions to type acts by means of *licensing conditions on type acts* which we can also refer to as action rules as discussed in Section 2.3. A basic licensing condition is that an agent can create (or contribute to the creation of) a witness for the first type that occurs on the agenda in its information state. Such a licensing condition is expressed in (53) where we use $s_{i,A}$ to represent the current information state of the agent A.

(53) $$\frac{s_{i,A} :_A \left[\text{agenda:}\begin{bmatrix}\text{fst:}\textit{RecType} \\ \text{rst:}\mathrm{list}(\textit{RecType})\end{bmatrix}\right]}{:_A\ s_{i,A}.\text{agenda.fst!}}$$

Update functions (or game rules) of the kind we have discussed are handled by the licensing conditions in (54), where we use f to represent an update function available to the agent, A, and e an event currently observed by the agent. As before $s_{i,A}$ refers to A's current information state and $s_{i+1,A}$ is used to refer to A's updated information state. We use e^* to refer to a current event.

(54) a. $$\frac{f : (T_1 \rightarrow (T_2 \rightarrow \mathit{Type})) \quad s_{i,A} :_A T_1 \quad e^* :_A T_2}{s_{i+1,A} :_A f(s_{i,A})(e^*)}$$

b. $$\frac{f : (T \rightarrow \mathit{Type}) \quad s_{i,A} :_A T}{s_{i+1,A} :_A f(s_{i,A})}$$

(54a) is for the case where the update function requires an event in order to be triggered and is thereby licensed (or "afforded") to judge their updated information state as being of the type resulting from applying the update function to their current information state and the observed event. (54b) is for the case where no event is required. We can think of updates to the information state licensed by this as *tacit* updates, that is, updates that do not require an external event such as a speech event or move in a game.

Licensing conditions will regulate the coordination of successfully realized games like fetch. They enable the agents to coordinate their activity when they both have access to the same objects of type *Game* and are both willing to play. The use of the word "license" is important, however. The agents have free will and may choose not to do what is licensed and also may perform acts that are not licensed. We cannot build a theory that will predict exactly what will happen but we can have a theory which tells us what kinds of actions belong to a game. It is up to the agents to decide whether they will play the game or not. At the same time, however, we might regard whatever is licensed at a given point in the game as an obligation. That is, if there is a general obligation to continue a game once you have embarked on it, then whatever type is placed on an agent's agenda as the result of a previous event in the game can be seen as an obligation on the agent to play its part in the creation of an event of that type.

2.4 Speech events

In Chapter 1 we talked about the perception of events such as a boy and a dog playing fetch. We imagined Kim walking through the park and perceiving various kinds of events. Suppose that she meets a friend in the park and they start to have a conversation. A conversation is a kind of event involving language which seems to be uniquely human. The kind of dialogue involved in a conversation enables humans to exchange information in a way that is more complex and more abstracted from currently occurring events than other animals seem capable of. Nevertheless, we will argue that the basic mechanisms of dialogue involve assigning types to events in way that we discussed in Chapter 1. The events involved are *speech events*.

Consider the kind of event type prediction that we considered in Chapter 1. Suppose that Kim sees the boy playing fetch with the dog and the boy is standing close to the lake with his back to it. As the dog runs towards him with the stick he takes a step backwards. "No," says Kim, seeing that the boy is about to fall in the lake. "Watch out," she shouts to the boy who takes a step forward just in time and narrowly misses falling in the lake. Her utterance of *no* represents a negative attitude towards a predicted outcome. This kind of negation is discussed briefly in Cooper and

Ginzburg (2011a,b) where examples are given of cases where *no* is a response to a completed event and where it is used as an attempt to prevent the predicted outcome. This latter exploits the fact that agents cannot only perceive and classify events according to the types to which they are attuned but can also intervene and prevent a predicted outcome. Kim's linguistic utterance of *watch out* is used in this way. While Kim is using words of English this is not yet completely linguistic interaction. A dog, sensing danger, will begin to bark and this can have the effect of preventing a predicted outcome. It is a kind of inter-agent communication nevertheless in that it is an intervention in the flow of events which involves predicting and changing the behaviour of another agent. In this sense it is similar to human dialogue, although human dialogue is normally a much more abstract affair, involving predicting and influencing the other agent's linguistic behaviour and the attitudes and beliefs which the other agent has concerning certain types.

Dialogues themselves are events and, just like other events, can be regarded as strings of smaller events. Consider the dialogue excerpt (55) from the British National Corpus which is the beginning of a consultation between a patient (John) and a doctor (Anon 1).

(55)	John:	Hello doctor.
	Anon 1:	Hello.
		Well Mr [last or full name], what can I [do for you today]$_1$?
	John:	[Er, it's]$_1$ a wee problem I've had for a ⟨pause⟩ say about a year now.
	Anon 1:	Mhm.
	John:	It's er my face.
		And my skin.
		I seem to get an awful lot of, it's like
	Anon 1:	Aha.
	John:	dry flaky skin.
	Anon 1:	Yeah.
	John:	And I get it on my forehead, [down here]$_2$
	Anon 1:	[I can see]$_2$

BNC file G43, sentences 1–13

We might assign the whole dialogue of which this is a part to a genre type for patient doctor consultation.[3] The genre type could be seen as an event type which, like the type for the game of fetch discussed in Chapter 1, can be broken down into a string of subevent types such as greeting (realized here by the exchange *Hello doctor./Hello*), establishing the patient's symptoms (realized here by the remainder of (55)), making a diagnosis, prescribing treatment and so on. These subevents can be further broken down into strings of turns which further can be broken down into strings of utterances of phrases. In turn phrase utterances are constituted by strings

[3]For a discussion of genre in the kind of framework that we are describing see Ginzburg (2010, 2012).

of word utterances which in turn can be regarded as strings of phoneme events. Notice that the temporal relationships between the elements of these strings is more varied than we accounted for in Chapter 1. In dialogue utterances may temporally overlap each other (as indicated in (55) by the notation $[\ldots]_n$). When we consider adjacent phoneme events in a string overlap becomes the norm (referred to as coarticulation in phonetics). Although we did not take it up in Chapter 1, temporal overlap in event strings is not restricted to speech events. For example, in the game of fetch it is quite often the case that the dog will start running after the stick before the human has finished throwing it. Perceiving temporally overlapping events is part of our basic perceptual apparatus.

We will work on developing a type for speech events, *SEvent*.[4] Crucial here is the type of phonological event, *Phon*, that is the type of event where certain speech sounds are produced. A field for events of this type will play a role corresponding to the phonology feature in HPSG (Sag *et al.*, 2003). For simplicity we might assume that *Phon* is an abbreviation for $\left[\text{e:}\textit{Word}\right]^{+}$ that is a non-empty string of events where a word is uttered.[5] Here *Word* is the type of event where word forms of the language are uttered. A more accurate proposal might be that *Phon* is $\left[\text{e:}\textit{Phoneme}\right]^{+}$[6] where *Phoneme* is the type of utterance event where a phoneme is uttered. This would still be a simplification and an abstraction from the actual events that are being classified, however. A phoneme type is rather to be regarded as a complex type of acoustic and articulatory event and what we regard as a string of phonemes is in fact a string of events where the phoneme types overlap (corresponding to what is know as *coarticulation* in phonological and phonetic theory). For example, the pronunciation of the phoneme /k/ in "kit" is distinct from its pronunciation in "cat" due to the influence of the following vowel. Suppose that the dimensions of phoneme utterance events are given by place, manner, rounding, voicing and nasality. Then we might represent the type of an utterance of /k/ as

(56) $$\begin{bmatrix} \text{place} & : & \textit{Velar} \\ \text{manner} & : & \textit{Stop} \\ \text{rounding} & : & \textit{NonRound} \\ \text{voicing} & : & \textit{NonVoiced} \\ \text{nasal} & : & \textit{NonNasal} \end{bmatrix}$$

the type of an utterance of /i/ by

[4]This type will be different for different languages, dialects, even idiolects. Thus there will be a different type corresponding to what we think of as speech events of English as opposed to speech events of French. Similar remarks can be made about all the linguistic types that we introduce. We will ignore this in our grammatical types in order to avoid proliferation of subscripts.

[5]If we want to be more grammatically sophisticated we might want to allow silent speech events by allowing empty phonologies, that is, we say that *Phon* is the type $\left[\text{e:}\textit{Word}\right]^{*}$.

[6]Or $\left[\text{e:}\textit{Phoneme}\right]^{*}$.

(57) $$\begin{bmatrix} \text{place} & : & \textit{FrontHigh} \\ \text{manner} & : & \textit{Vocalic} \\ \text{rounding} & : & \textit{NonRound} \\ \text{voicing} & : & \textit{Voiced} \\ \text{nasal} & : & \textit{NonNasal} \end{bmatrix}$$

and the type of an utterance of /æ/ by

(58) $$\begin{bmatrix} \text{place} & : & \textit{BackHigh} \\ \text{manner} & : & \textit{Vocalic} \\ \text{rounding} & : & \textit{NonRound} \\ \text{voicing} & : & \textit{Voiced} \\ \text{nasal} & : & \textit{NonNasal} \end{bmatrix}$$

Naively, one might think that the type of the phoneme string /ki/ would be

(59)
$$\begin{bmatrix} \text{e} & : & \begin{bmatrix} \text{place} & : & \textit{Velar} \\ \text{manner} & : & \textit{Stop} \\ \text{rounding} & : & \textit{NonRound} \\ \text{voicing} & : & \textit{NonVoiced} \\ \text{nasal} & : & \textit{NonNasal} \end{bmatrix} \end{bmatrix} \frown \begin{bmatrix} \text{e} & : & \begin{bmatrix} \text{place} & : & \textit{FrontHigh} \\ \text{manner} & : & \textit{Vocalic} \\ \text{rounding} & : & \textit{NonRound} \\ \text{voicing} & : & \textit{Voiced} \\ \text{nasal} & : & \textit{NonNasal} \end{bmatrix} \end{bmatrix}$$

However, the place of articulation of the /k/ will be influenced by the place of articulation of the following vowel as in (60)

(60)
$$\begin{bmatrix} \text{e} & : & \begin{bmatrix} \text{place} & : & \textit{Palatal} \\ \text{manner} & : & \textit{Stop} \\ \text{rounding} & : & \textit{NonRound} \\ \text{voicing} & : & \textit{NonVoiced} \\ \text{nasal} & : & \textit{NonNasal} \end{bmatrix} \end{bmatrix} \frown \begin{bmatrix} \text{e} & : & \begin{bmatrix} \text{place} & : & \textit{FrontHigh} \\ \text{manner} & : & \textit{Vocalic} \\ \text{rounding} & : & \textit{NonRound} \\ \text{voicing} & : & \textit{Voiced} \\ \text{nasal} & : & \textit{NonNasal} \end{bmatrix} \end{bmatrix}$$

In addition to this the voice onset associated with the vowel will normally begin before the articulation of the stop is complete as in (61).

(61)
$$\left[\begin{array}{lll} \mathrm{e} & : & \left[\begin{array}{lll} \text{place} & : & \textit{Palatal} \\ \text{manner} & : & \textit{Stop} \\ \text{rounding} & : & \textit{NonRound} \\ \text{voicing} & : & \textit{NonVoiced}^\frown\textit{Voiced} \\ \text{nasal} & : & \textit{NonNasal} \end{array}\right] \end{array}\right]^\frown\left[\begin{array}{lll} \mathrm{e} & : & \left[\begin{array}{lll} \text{place} & : & \textit{FrontHigh} \\ \text{manner} & : & \textit{Vocalic} \\ \text{rounding} & : & \textit{NonRound} \\ \text{voicing} & : & \textit{Voiced} \\ \text{nasal} & : & \textit{NonNasal} \end{array}\right] \end{array}\right]$$

This is not meant to be a serious phonological analysis. We include it here to show how the well-studied phenomenon of coarticulation could be included in the general framework and to show that the notion of overlapping events which we will need later for semantics and dialogue is the same notion that is needed for phonology. We have no more to say about phonology and will limit our analysis of phonological events to strings of words.

We will keep the simplifying assumption that phonology is a string of words here (that is, that *Phon* is *Word*$^+$ and we do not say more about what is of type *Word*) as we do not aim to give a detail account of phonology. Thus a proposal for the type *SEvent* might be (62).

(62) $\left[\begin{array}{lll} \mathrm{e} & : & \textit{Phon} \end{array}\right]$

To this we might usefully add the speech location as in (63).

(63) $$\left[\begin{array}{lll} \text{e-loc} & : & \textit{Loc} \\ \mathrm{e} & : & \textit{Phon} \\ \mathrm{c_{loc}} & : & \text{loc(e,e-loc)} \end{array}\right]$$

We will take *Loc* to be the type of regions in three dimensional space without specifying more detail. Further if e is an event and l a location we will say that the type loc(e,l) is non-empty just in case e is located at l, again without saying exactly what that means for now.

It might seem natural to add roles of speaker and audience, given what we know about speech act theory (Searle, 1969). Thus we might consider *SEvent* to be the type in (64).

(64) $$\left[\begin{array}{lll} \text{e-loc} & : & \textit{Loc} \\ \text{sp} & : & \textit{Ind} \\ \text{au} & : & \textit{Ind} \\ \mathrm{e} & : & \textit{Phon} \\ \mathrm{c_{loc}} & : & \text{loc(e,e-loc)} \\ \mathrm{c_{sp}} & : & \text{speaker(e,sp)} \\ \mathrm{c_{au}} & : & \text{audience(e,au)} \end{array}\right]$$

However, while many speech events may be considered to be of this type, not all will. Of course, some speech events are not addressed to any audience. An example might be an exclamation uttered after hitting one's thumb with a hammer. Longer speech events like dialogues will not have a single speaker or audience. Even shorter chunks corresponding perhaps to single speech acts do not always have a single speaker or audience. For example, consider split utterances as discussed by Purver *et al.* (2010) who give the example (65).

(65) A: I heard a shout. Did you
B: Burn myself? No, luckily.

Here we probably want to consider the utterance of *Did you . . . burn myself?* as a speech event on which A and B collaborate. Otherwise it might be hard to explain how *you* can be interpreted as the subject of *burn*. We have a single predication split across two speakers. Similarly, speakers can address different audiences within the same predicate structure as in (66).

(66) You [pointing] work with you [pointing] and you [pointing] work on your own.

Nevertheless, we might consider that the majority of speech events would belong to the more restricted type (64).

Because we have taken a neo-Davidsonian (Dowty, 1989) approach to the more restricted speech-event types, where the objects playing the various roles in the speech events are introduced in separate fields, both (63) and (64) are subtypes of (62). We will use *SEvent* below to represent the most specific of the types, (64), while bearing in mind that many events we may want to call "speech events" will belong only to more general types such as (63) and (62).

2.5 Signs

We interpret many speech events as being associated with a semantic content, but not all. When John in (55) says *It's a wee problem I've had for a, say, about a year now* he is using the speech event to refer to another situation - a situation in which he has dry skin for a period of a year. This is what Barwise and Perry (1983) would refer to as the *described situation* which is distinct from the speech situation. In contrast the doctor's utterance of *Hello* in (55) does not tell us anything about a described situation external to the current conversation, although it does give us information about where we are in the conversation (the beginning) and indicate that the doctor is paying attention. We shall say that the former utterance is associated with a type of described situation and call this the *content* of the utterance. A situation type is an appropriate content for a declarative sentence used to make an assertion.[7] The contents of phrases within such a sentence

[7]We will discuss later that alternative proposed in Ginzburg (2012) that it should be a pairing of a situation type with a situation, that is an Austinian proposition as introduced by Barwise and Perry (1983) based on Austin (1961).

such as *a wee problem* or *about a year* will be objects which can be combined to produce such a type. The contents of other kinds of speech acts, for example, associated with questions like the doctor's utterance of *what can I do for you today?* will be objects based on situation types, in the case of this question a function which maps actions to a situation type. (See Ginzburg (2012) for a discussion of the kind of treatment of questions we have in mind.)

We can think of this association of content with a speech event in similar terms to prediction of event completion discussed in Section 2.2 of Chapter 1. At least in the case of declarative assertions it is a mapping from an observation of a situation to a type of situation. In the case of the event completion the result of the mapping was a type for the completion of the event so far observed. In the case of the speech event we are relating the observation to a type of situation which is entirely distinct from the speech event. The association is less immediate and more abstract but the underlying mechanism, associating the observation of a situation of a given type with another type and drawing the conclusion that the second type must be non-empty, is the same. We could represent the association by a function of the form (67), corresponding to (20) in Chapter 1.

(67) $\lambda s : T_{SpEv} \,.\, T_{Cont}(s)$

This represents a mapping from a speech event s of a given type T_{SpEv} to a type T_{Cont} which is the content of the speech event. The type T_{Cont} can depend on s (for example, the type of the described situation may require that the described situation be related to the utterance situation temporally or spatially).

de Saussure (1916) called the association between speech and content a sign and this notion has been taken up in modern linguistics in Head Driven Phrase Structure Grammar (HPSG, Sag *et al.*, 2003). In HPSG a sign is regarded technically as a feature structure and our notion of record type correponds to a feature structure. One way in which our type system differs from HPSG is that we have both records and record types where HPSG has just feature structures. We will consider a sign to be a record representing a pairing of a speech event and a type representing the content. One advantage of considering a sign as a record rather than a function as in (67) is that there is no directionality in a record as there is in a function. Thus the record can be associated with either interpretation (from speech event to content) or generation (from content to speech event). We can make a straightforward relationship between a function such as (67) and a record type (68).

(68) $\left[\begin{array}{lcl} \text{s-event} & : & T_{SpEv} \\ \text{cont=}T_{Cont} & : & \mathit{Cont} \end{array}\right]$

(68) is a type of signs. Notice that the 'cont'-field in (68) is a manifest field corresponding to the fact that the function in (67) returns the type T_{Cont}, not an object of type T_{Cont}. This means

that the 'cont' field in (68) requires that the type itself is in the 'cont' field in a record of the type, that is, in the sign. The type *Cont* is the type of contents. For the moment we will say that *Cont* is the type *RecType*, that is, that contents are record types. This is because, for the moment, we will restrict our attention to declarative sentences. When we come to look at constituents of sentences and speech acts other than assertions we will need to expand *Cont* to include other kinds of entities as well. Restricting our attention first to complete declarative sentences is similar to starting with propositional logic before moving on to more complex analysis. The type *Sign* of signs in general is given in (69).

(69) $\begin{bmatrix} \text{s-event} & : & \mathit{SEvent} \\ \text{cont} & : & \mathit{Cont} \end{bmatrix}$

A record of this type, a sign, will pair a speech event with a content. We will refine our definition of *Sign* as we progress. We will often have occasion to use subtypes of *Sign* to characterize the signs used in our analyses. We will call these *sign types* and introduce the type *SignType* whose witnesses we characterize in (70).

(70) T : *SignType* iff $T \sqsubseteq$ *Sign*

Sign types are a particular kind of record type so we will also have (71).

(71) *SignType* $\sqsubseteq$ *RecType*

2.6 Information exchange in dialogue

We start by considering simple dialogues such as (72) which might occur between two people one of whom is instructing the other about simple facts or between a user and a system where the user is adding simple facts to a database using a natural lanuage interface.

(72)

(72)	User:	Dudamel is a conductor
	System:	Aha
	User:	Beethoven is a composer
	System:	OK

The job of the dialogue partner identified as "System" is to record the facts in memory and confirm to the dialogue partner identified as "User" that this has happened. It seems straightforward to think of the user's utterances in (72) as corresponding to signs as described in Section 2.5. For example, the user's first utterance could be regarded as corresponding to a sign of the type in (73).

(73) $\left[\begin{array}{lll} \text{s-event:}\left[\begin{array}{lll}\text{e} & : & \text{“Dudamel is a conductor”}\end{array}\right] & & \\ \text{cont=}\left[\begin{array}{lll}\text{e} & : & \text{conductor(dudamel)} \\ \text{c}_{\text{tns}} & : & \text{final\_align}(\Uparrow\text{s-event.e,e})\end{array}\right] & : & \textit{RecType}\end{array}\right]$

Here “Dudamel is a conductor” is a convenient abbreviation for (74).

(74) $\left[\text{e:“Dudamel”}\right]^\frown\left[\text{e:“is”}\right]^\frown\left[\text{e:“a”}\right]^\frown\left[\text{e:“conductor”}\right]$

where for any word w, “w” is the type of event where w is uttered. “Dudamel is a conductor” is thus a type of string of events of word utterances and is thus a subtype of *Phon*, given our assumptions in Section 2.4.

The content is that Dudamel is a conductor and that his being a conductor is aligned with the speech event in that the speech event occurs simultaneously with the end of the event of Dudamel being a conductor. This is not to say that Dudamel will not continue to be a conductor after the speech event but rather to say that we are aligning the speech event with what has happened so far up to and including the speech event. (The simple present in English in contrast to the present progressive and the simple present in many other languages seems to require this.) How do we align events? We use the technique developed by Fernando (see, for example, Fernando, 2008) of creating a single event which includes both events as a part. We will exploit our record technology to keep track of the separate events in the larger event and to achieve something corresponding to what Fernando calls *superposition*. We might require that the event which is the coordination of the two events of type “Dudamel is a conductor” and ‘conductor(dudamel)’ is of the type in (75).

(75) $\left[\begin{array}{lll}\text{e}_1 & : & \text{“Dudamel is a conductor”} \\ \text{e}_2 & : & \text{conductor(dudamel)}\end{array}\right]$

Another option is to require that the coordinated event type explicitly allow for there to be events of the type ‘conductor(dudamel)’ prior to the utterance as in (76).

(76) $\left[\begin{array}{lll}\text{e} & : & \text{conductor(dudamel)}\end{array}\right]^{*\frown}\left[\begin{array}{lll}\text{e} & : & \left[\begin{array}{lll}\text{e}_1 & : & \text{“Dudamel is a conductor”} \\ \text{e}_2 & : & \text{conductor(dudamel)}\end{array}\right]\end{array}\right]$

Here the dimension ‘e’ splits into two subdimension ‘e.e$_1$’ and ‘e.e$_2$’. If we wish to be explicit about the fact that a situation of type “Dudamel is a conductor” is a string of word utterances we can give the more detail type in (77).

(77) $\left[\ \text{e} \ : \ \text{conductor(dudamel)}\ \right]^{*\frown} \left[\ \text{e} \ : \ \left[\begin{array}{lll} \text{e}_1 & : & \text{“Dudamel”} \\ \text{e}_2 & : & \text{conductor(dudamel)} \end{array}\right]\ \right]^{\frown}$
$\left[\ \text{e} \ : \ \left[\begin{array}{lll} \text{e}_1 & : & \text{“is”} \\ \text{e}_2 & : & \text{conductor(dudamel)} \end{array}\right]\ \right]^{\frown}$
$\left[\ \text{e} \ : \ \left[\begin{array}{lll} \text{e}_1 & : & \text{“a”} \\ \text{e}_2 & : & \text{conductor(dudamel)} \end{array}\right]\ \right]^{\frown}$
$\left[\ \text{e} \ : \ \left[\begin{array}{lll} \text{e}_1 & : & \text{“conductor”} \\ \text{e}_2 & : & \text{conductor(dudamel)} \end{array}\right]\ \right]$

This explicitly requires that Dudamel is a conductor during the utterance of each individual word. Both the types (76–77) are facilitated by the fact that ‘conductor(dudamel)’ is a *state*-type, that is, given a situation e : conductor(dudamel) we can regard it as a string of events of type $\left[\text{e:conductor(dudamel)}\right]^{+}$. We will return to aspectual types other than state below. The predicate ‘final_align’ in (73) requires alignment of the speech event and the described event in the way we have exemplified in (76) and (77).

If s is a string, we use the notation $s[i]$ to represent the object which is in the ith position of s, that is, in the records that we use to code strings, $s.t_i$. $s[0]$ represents the first element in s. The definition of what counts as a witness for final_align(e_1,e_2), given in (78) and repeated in Appendix 19, requires that e is of this type just in case e is an event where e_1 is aligned with a final segment of e_2, that is in e there is a split in dimension in the final segment as illustrated in (77).

(78) If s_1:*Rec*$^+$ is a string of length n and s_2:*Rec*$^+$ is a string of length m, then s : final_align(s_1,s_2) iff

1. m is greater than or equal to n
2. s is a string of length m
3. for each i, $0 \leq i < n$,
   a) $s[(m-n)+i] : \begin{bmatrix} \text{e}_1\text{:}\textit{Rec} \\ \text{e}_2\text{:}\textit{Rec} \end{bmatrix}$
   b) $s[(m-n)+i]$.e$_1$ = $s_1[i]$
   c) $s[(m-n)+i]$.e$_2$ = $s_2[(m-n)+i]$
4. otherwise for each i, $0 \leq i < m$, $s[i] = s_2[i]$

Suppose that s_1 and s_2 are the two strings in (79) of lengths 4 and 5 respectively.

(79) a. $s_1[0]s_1[1]s_1[2]s_1[3]$

b. $s_2[0]s_2[1]s_2[2]s_2[3]s_2[4]$

Then s will be the string in (80).

(80) $$s_2[0]\begin{bmatrix} \text{e}_1 & = & s_1[0] \\ \text{e}_2 & = & s_2[1] \end{bmatrix}\begin{bmatrix} \text{e}_1 & = & s_1[1] \\ \text{e}_2 & = & s_2[2] \end{bmatrix}\begin{bmatrix} \text{e}_1 & = & s_1[2] \\ \text{e}_2 & = & s_2[3] \end{bmatrix}\begin{bmatrix} \text{e}_1 & = & s_1[3] \\ \text{e}_2 & = & s_2[4] \end{bmatrix}$$

In terms of our informal proof theoretic notation we might write (81).

(81) For Γ with string types

a. $$\frac{\begin{array}{c} \Gamma \vdash s_1 : \textit{Rec}^n \\ \Gamma \vdash s_2 : \textit{Rec}^m \\ \Gamma \vdash s : \textit{Rec}^m \end{array} \qquad m \geq n \qquad \begin{array}{c} [0 \leq i < n] \\ \vdots \\ \Gamma \vdash s[(m-n)+i] = \begin{bmatrix} \text{e}_1 & = & s_1[i] \\ \text{e}_2 & = & s_2[(m-n)+i] \end{bmatrix} \end{array}}{\Gamma \vdash s : \text{final\_align}(s_1, s_2)}$$

b. $$\frac{\begin{array}{c} \Gamma \vdash s_1 : \textit{Rec}^n \\ \Gamma \vdash s_2 : \textit{Rec}^m \\ \Gamma \vdash s : \textit{Rec}^m \end{array} \qquad \begin{array}{c} m \geq n \\ 0 \leq i < n \end{array} \qquad \Gamma \vdash s : \text{final\_align}(s_1, s_2)}{\Gamma \vdash s[(m-n)+i] = \begin{bmatrix} \text{e}_1 & = & s_1[i] \\ \text{e}_2 & = & s_2[(m-n)+i] \end{bmatrix}}$$

The notation '⇑' in (73) indicates that the path 'x' is not to be found in the local record type which is required to be the value of 'cont' but in the next higher record type with the fields 's-event' and 'cont'.

(73) is a more friendly notation for (82).

(82) $$\begin{bmatrix} \text{s-event:} \begin{bmatrix}\text{e:“Dudamel is a conductor”}\end{bmatrix} \\ \text{cont:}\langle \lambda v_1\text{:“Dudamel is a conductor” . } \textit{RecType}_{\begin{bmatrix}\text{e:conductor(dudamel)} \\ \text{c}_{\text{tns}}\text{:}\langle \lambda v_2\text{:conductor(dudamel) . final\_align}(v_1, v_2), \langle \text{e} \rangle\rangle \end{bmatrix}}, \langle \text{s-event.e} \rangle\rangle \end{bmatrix}$$

The '⇑' notation is inspired by the use of '⇑' in Lexical Function Grammar (Dalrymple *et al.*, 1995) pointing up a hierarchical structure.

An alternative involves unique identifiers and is inspired by the notation used for reentrancy in Head-Driven Phrase Structure Grammar (Sag *et al.*, 2003). The example in this notation is given in (83).

(83) $$\left[\begin{array}{l} \text{s-event:}\left[\begin{array}{lll} \text{e}_{\boxed{0}} & : & \text{“Dudamel is a conductor”} \end{array}\right] \\ \text{cont=}\left[\begin{array}{lll} \text{e}_{\boxed{1}} & : & \text{conductor(dudamel)} \\ \text{c}_{\text{tns}} & : & \text{final\_align}(\boxed{0},\boxed{1}) \end{array}\right]:\textit{RecType} \end{array}\right]$$

We will call this last notation *unique identifier notation*. We will go into this notation with some rigour because it will be useful later when we introduce relabelling of record types in Chapter 4. The unique identifier notation for a type, T, can be obtained from the official notation for T by carrying out in order the manipulations characterized in (84), repeated in Appendix 13.

(84)
1. Add a unique identifier $\boxed{i}$ (where i is a natural number) to the last label occurrence in any path, π, which is referenced in a dependent field, $\langle f, \langle \ldots, \pi, \ldots \rangle\rangle$, somewhere in T.
2. Replace π in any dependent field, $\langle f, \langle \ldots, \pi, \ldots \rangle\rangle$, with the unique identifier associated with the final label occurrence in π.
3. Replace any pair in a dependent field of the form

   $\langle \lambda v_1 : T_1 \ldots \lambda v_n : T_n \,.\, \varphi(\!(v_1, \ldots, v_n)\!), \langle \boxed{i_1}, \ldots, \boxed{i_n} \rangle\rangle$

   with

   $\varphi(\!(\boxed{i_1} : T_1, \ldots, \boxed{i_n} : T_n)\!)$

The final step involves a variant of β-conversion. It is important to represent the type restriction associated with the unique identifier by the domain type of the function since the domain type of the function may be distinct from the type associated with the identifier in the field labelled by the identifier. However, we suppress the type when the domain type of the function is identical with the type associated with the path in T which is referenced (as is most often the case in the examples we discuss). This is illustrated in (83).

We also show how to convert back from unique identifier notation for a type, T, to official notation. We can do this by carrying out the manipulations characterized in (85) in order (repeated in Appendix 13)

(85)
1. For each occurrence of $\boxed{i}$ in a dependent field labelled ℓ_d (the label for the dependent field) for some natural number, i, locate the smallest record structure, Δ, in T which contains a path, π_d, to ℓ_d and a path π_a to a label $\ell_{\boxed{i}}$ (the label for the addressed field)

2. Let ℓ be the first label on π_d. If $\Delta.\ell$ is an ordered pair $\langle\lambda v_j{:}T_j\,.\,\varphi((\boxed{i}{:}T_i)), \langle\pi_1,\ldots,\pi_n\rangle\rangle$, then replace $\Delta.\ell$ with $\langle \lambda v_i{:}T_i\,.\,\lambda v_j{:}T_j\,.\,\varphi[\boxed{i}{:}T_i \rightsquigarrow v_i], \langle\pi_1,\ldots,\pi_n,\pi_a\rangle\rangle$. Otherwise, if $\Delta.\ell$ is $\varphi((\boxed{i}{:}T_i))$, replace $\Delta.\ell$ with $\langle\lambda v_i{:}T_i\,.\,\varphi[\boxed{i}{:}T_i\rightsquigarrow v_i], \langle\pi_a\rangle\rangle$
3. For any number i, remove the subscript $\boxed{i}$ on any label on which it occurs.

The sign type (73) seems to give us what we need in order to explain how an utterance of *Dudamel is a conductor* can convey the information that Dudamel is a conductor. If both dialogue participants have this sign type among their resources then the User knows that in order to convey this content she has to make an utterance which witnesses the appropriate speech event type. The System knows that on observing a speech event of this type the corresponding content should be recorded. Simple action rules relating to the sign type (73)/(85) are given in (86) to illustrate the kind of reasoning we are thinking of. These rules assume that the agent A has access to the type (73)/(85) as a resource, either stored in memory or available through computation using other accessible resources.

(86) a.

$$\frac{u :_A \text{“Dudamel is a conductor”}}{:_A \begin{bmatrix} \text{s-event:} & \begin{bmatrix}\text{e}{=}u\text{:“Dudamel is a conductor”}\end{bmatrix} \\ \text{cont=} & \begin{bmatrix}\text{e:conductor(dudamel)}\\ \text{c}_{\text{tns}}\text{:final\_align}(\Uparrow\text{s-event.e,e})\end{bmatrix}\text{:}\mathit{RecType}\end{bmatrix}}$$

b.

$$\frac{s_{i,A} :_A \begin{bmatrix} \text{agenda:} & \begin{bmatrix} \text{fst=} & \begin{bmatrix} \text{s-event:} & \begin{bmatrix}\text{e:}\mathit{Phon}\end{bmatrix} \\ \text{cont=} & \begin{bmatrix}\text{e:conductor(dudamel)}\\ \text{c}_{\text{tns}}\text{:final\_align}(\Uparrow\text{s-event.e,e})\end{bmatrix}\text{:}\mathit{RecType}\end{bmatrix}\text{:}\mathit{RecType} \\ \text{rst:list}(\mathit{RecType}) & \end{bmatrix}\end{bmatrix}}{s_{i+1,A} :_A \begin{bmatrix} \text{agenda:} & \begin{bmatrix} \text{fst=} & \begin{bmatrix} \text{s-event:} & \begin{bmatrix}\text{e:“Dudamel is a conductor”}\end{bmatrix} \\ \text{cont=} & \begin{bmatrix}\text{e:conductor(dudamel)}\\ \text{c}_{\text{tns}}\text{:final\_align}(\Uparrow\text{s-event.e,e})\end{bmatrix}\text{:}\mathit{RecType}\end{bmatrix}\text{:}\mathit{RecType} \\ \text{rst=}s_{i,A}\text{.agenda.rst:list}(\mathit{RecType}) & \end{bmatrix}\end{bmatrix}}$$

(86a) tells us that if an agent, A, judges an utterance, u, to be of the phonological type we are representing as “Dudamel is a conductor” then A is licensed to make the judgement that there is a sign, that is, an event, of the type (73)/(86) of which u is the speech component. (86b) tells us that if an agent, A, is planning to make an utterance with the content that Dudamel is a conductor, that is, there is a sign type on A’s agenda which specifies the content but not specify the exact nature of the phonological event required, then A is licensed to update her information state with the type (73)/(86) which does specify the type of the phonological content and has the same content. The rules in (86) are, of course, very specific and rely on a simple notion of information state in which there is an agenda. In what follows we will try to move towards more general rules based on sign types which can be derived by parsing and generation techniques and compositional processing of speech events. However, these rules illustrate how we can relate the kind of abstract and static sign types that we are proposing to dynamic actions afforded by given events and information states of an agent.

Things are not as straightforward, however, for the acknowledgements *Aha* and *OK* expressed by the system. It is not obvious whether these utterances are to be regarded as signs at all. Certainly a speech event is involved but one might question what content they have. One suggestion would be that the content of *Aha* uttered after an assertion by the other dialogue partner would be the same as the content of that assertion. Thus the system is expressing the same content as the user. This may or may not be true. But such an analysis seems to be missing a central point about what is going on in this dialogue, namely that the user is making an assertion and the system is acknowledging that the content has been accepted and duly processed. In order to account for this kind of fact Ginzburg in a large body of work has developed the notion of a dialogue gameboard, most recently formulated in terms of TTR in Ginzburg (2012); Ginzburg and Fernández (2010). In the computational dialogue systems literature this have given rise to the Information State Update (ISU) approach (Larsson and Traum, 2001; Larsson, 2002) which is also described in Ginzburg and Fernández (2010). In Chapter 1 we introduced the notion of an information state as a record containing a field labelled ‘agenda’ and used the word “gameboard” to refer to a type of information state. Our aim there was to show that the kind of gameboard analysis introduced for dialogue in this literature is also important for the coordination of joint action by agents in general. The gameboards that have been used for dialogue analysis have a number of fields in addition to the agenda. Each dialogue participant will have among their resources a record type, their dialogue gameboard which represents their understanding of (what Larsson call their take on) their current information state. Following Larsson (2002) we place information which the agent assumes to be common with its interlocutors under the label ‘shared’ in the gameboard and also have a field with the label ‘private’ representing information about the state of the dialogue which is not shared with other dialogue participants. This will include, for example, plans for what should be said next represented in the agenda. In Figure 2.3 we give a schematic view of the gameboards associated with each of the dialogue participants in the first exchange in (72).

This assumes ideal communication. There is lots that could go wrong which could have the consequence that the two agents become misaligned and an important part of this framework is to provide a basis for the description of miscommunication as well as communication. (See Ginzburg (2012) for more discussion of this.)

We treat the dialogue information states represented by the square boxes as records as in (87).

(87) $$\left[\begin{array}{lll} \text{private} & = & \left[\begin{array}{lll} \text{agenda} & = & \mathit{AGENDA} \end{array}\right] \\ \text{shared} & = & \left[\begin{array}{lll} \text{latest-utterance} & = & \mathit{L\text{-}UTT} \\ \text{commitments} & = & \mathit{COMM} \end{array}\right] \end{array}\right]$$

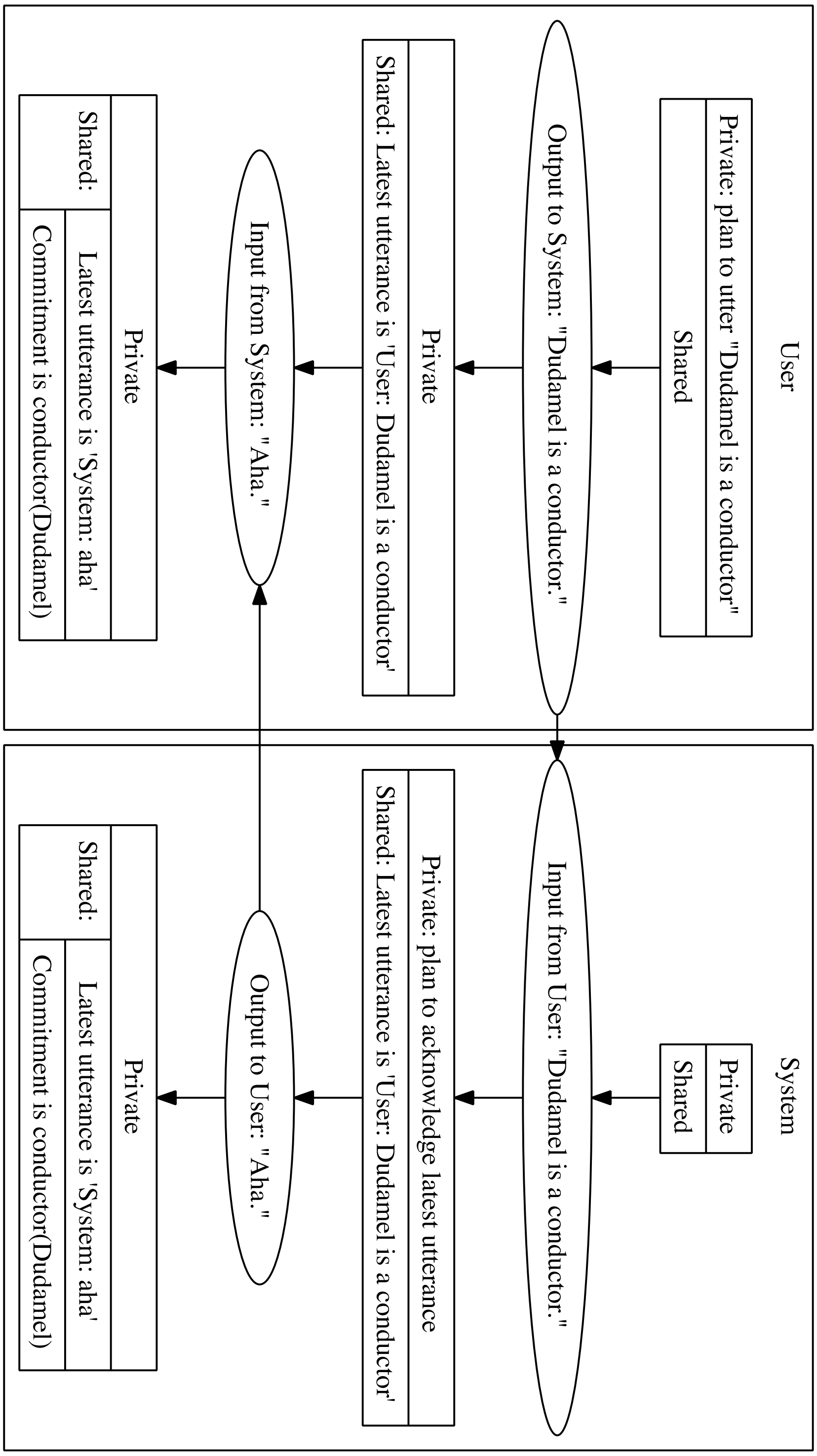


Figure 2.3: Dialogue management:"Dudamel is a conductor"

What kinds of objects should *AGENDA*, *L-UTT* and *COMM* be?

We will say that *AGENDA* is a list of sign types, that is, the types of sign that the agent plans to realize by means of a creation type act. Recall from Chapter 1 that this does not necessarily mean that the agent is the main actor in the event realizing the sign type. It can for example be a type of move to be carried out by an interlocutor which the agent should wait for. This will give us a mechanism for handling basic turn-taking in dialogue. (See Sacks *et al.*, 1974 for the classic work on turn-taking.) For now we will say that there are two ways in which an agent can be involved in a dialogue act: as speaker (or performer) or as hearer (part of the audience to whom the dialogue act is addressed).[8]

L-UTT should tell us what the latest utterance in the dialogue was. This will be the witness of a sign type.

The commitments field has normally been considered as a set of facts or propositions (Ginzburg, 2012; Larsson, 2002). Here we will treat them as a single record type, i.e. a witness of the type *RecType*. Using a single type will make it more straightforward to deal with issues like consistency and anaphora as we will see in later chapters.

Thus information states can belong to the type (88), our current version of the type *InfoState*.

(88) $$\left[\begin{array}{lll} \text{private} & : & \left[\begin{array}{lll} \text{agenda} & : & \text{list}(\textit{RecType}) \end{array}\right] \\ \text{shared} & : & \left[\begin{array}{lll} \text{latest-utterance} & : & \textit{Sign}^* \\ \text{commitments} & : & \textit{RecType} \end{array}\right] \end{array}\right]$$

Here *Sign** is the type of strings of signs of length 0 or more using the string type notation introduced on p. 46. At the beginning of a dialogue there will be no latest utterance and we will represent this by having the empty string of signs in 'latest-utterance'-field. By using *Sign** we allow for the possibility that the previous utterance can be represented by a string of several signs although in the examples we will discuss here we will only have strings of length 1.

At the beginning of a dialogue there will not be any shared commitments either. Therefore, it will be natural to use *Rec* for the commitments at the beginning of a dialogue. *Rec* is the type of all records. If we think of records as modelling situations then a commitment represented by *Rec* is a commitment to the existence of a situation but not to a situation of any particular type. Thus it corresponds to "there is a situation" or "the world is not empty". It plays a similar role in our theory to the set of all possible worlds in a system based on possible worlds. It represents a state where no constraints have been placed on the nature of the world. The type *Rec* is one of the witnesses of *RecType* (see Appendix 11.2). The type of an initial information state based on (88) is (89), that is, our current version of the type *InitInfoState*.

[8]A third way of being involved in a dialogue act which we will not take account of here is as an overhearer.

(89) $$\left[\begin{array}{lll} \text{private} & : & \left[\begin{array}{lll} \text{agenda=[ ]} & : & \text{list}(\mathit{RecType}) \end{array}\right] \\ \text{shared} & : & \left[\begin{array}{lll} \text{latest-utterance=}\varepsilon & : & \mathit{Sign}^* \\ \text{commitments=}\mathit{Rec} & : & \mathit{RecType} \end{array}\right] \end{array}\right]$$

Some signs (but not all) will be associated with an *illocutionary force*, a term which originally comes from Austin (1962). The four illocutionary forces we will consider here are: assertion, query, command and acknowledgement. Signs which have an illocutionary force can be thought of as *dialogue moves* or in Austin's original terminology *illocutionary acts*. Those signs which are not associated with an illocutionary force are normally constituents of something which does have illocutionary force. Thus, for example, if somebody says *The dog barked* the whole utterance can be thought of as an assertion. However, the utterance of *the dog* which is part of this utterance does not have illocutionary force. This is not to say that some other utterance of *the dog* could not have illocutionary force. For example, in response to the question *What made all the mess?*, an utterance of *the dog* might be regarded as an assertion that the dog made all the mess.

We introduce four subtypes of *Sign*: *Assertion*, *Query*, *Command* and *Acknowledgement*. These are characterized in (90).

(90) $$\begin{array}{lll} \mathit{Assertion} & – & \left[\begin{array}{lll} \text{s-event} & : & \mathit{SEvent} \\ \text{cont} & : & \mathit{RecType} \\ \text{illoc} & : & \text{assert(s-event, cont)} \end{array}\right] \\ \mathit{Query} & – & \left[\begin{array}{lll} \text{s-event} & : & \mathit{SEvent} \\ \text{cont} & : & \mathit{Question} \\ \text{illoc} & : & \text{query(s-event, cont)} \end{array}\right] \\ \mathit{Command} & – & \left[\begin{array}{lll} \text{s-event} & : & \mathit{SEvent} \\ \text{cont} & : & \mathit{RecType} \\ \text{illoc} & : & \text{command(s-event, cont)} \end{array}\right] \\ \mathit{Acknowledgement} & – & \left[\begin{array}{lll} \text{s-event} & : & \mathit{SEvent} \\ \text{cont} & : & \mathit{RecType} \\ \text{illoc} & : & \text{acknowledge(s-event, cont)} \end{array}\right] \end{array}$$

Note that the type of the content varies with the illocutionary force. We use the type *Question* for queries and *RecType* for the others. We will say more about *Question* in later chapters. It is quite likely that the content type for commands should be something other than *RecType*, for example, the type *Ppty* ("property") that we will develop in later chapters, but we do not have more to say about commands in this work. In a more complete treatment of illocutionary force the nature of the speech event could also be made to vary with illocutionary force. For example, we could require question syntax for queries, although that would not take account of the fact that declarative sentence syntax can also be used to ask questions. It is not our aim here to give a detailed analysis of such phenomena but to provide a general framework in which they could be analyzed.

We will also introduce types *AssertionType*, *QueryType*, *CommandType* and *Acknowledgement-Type* which are characterized in a similar way to *SignType* as in (91).

(91) a. T : *AssertionType* iff $T \sqsubseteq$ *Assertion*

b. T : *QueryType* iff $T \sqsubseteq$ *Query*

c. T : *CommandType* iff $T \sqsubseteq$ *Command*

d. T : *AcknowledgementType* iff $T \sqsubseteq$ *Acknowledgement*

In order to find the content of an utterance of *ok*, we look to the content of the previous utterance. Thus an utterance of *ok* following an utterance of *Dudamel is a conductor* will have the same content as the assertion, namely (92).

(92) $\left[\begin{array}{lll} \text{e} & : & \text{conductor(dudamel)} \end{array}\right]$

Assigning (92) as the content of *Dudamel is a conductor* involves the naive assumption that a proper name uniquely identifies a particular individual. We will develop a more sophisticated approach to proper names in Chapters 3 and 4.

Let us consider the update function and update rule which the user could use in order to update her information state after her own utterance of *Dudamel is a conductor*. This is modelled on the kind of integration rules discussed in Larsson (2002). The update function we wish to characterize will be defined on information states which have some assertion type as the first element on the agenda, that is the type in (93).

(93) $$\left[\begin{array}{lll} \text{private} & : & \left[\begin{array}{lll} \text{agenda} & : & \left[\begin{array}{lll} \text{fst} & : & \textit{AssertionType} \\ \text{rst} & : & \text{list}(\textit{RecType}) \end{array}\right] \end{array}\right] \\ \text{shared} & : & \left[\begin{array}{lll} \text{latest-utterance} & : & \textit{Sign}^* \\ \text{commitments} & : & \textit{RecType} \end{array}\right] \end{array}\right]$$

Rather than writing out such types in our update functions we can think of the objects in the domain of our update function as meeting the requirement that they belong to two types as in (94).

(94) a. *InfoState*

b. $\left[\text{ private } : \left[\text{ agenda } : \left[\text{ fst } : \textit{AssertionType}\ \right]\ \right]\ \right]$

A statement of the update function in terms of the abbreviation *InfoState* will not only save space but also be helpful as we further develop our notion of what type *InfoState* represents. If we are just adding extra fields to the type *InfoState*, that is we develop our theory monotonically just by adding more detail, then we do not have to go back and revise the formulation of the update functions based on earlier versions of the theory.

However, we still need to give a characterization of a single type which will serve as the domain type of the update function. One way to do this is to use TTR's *meet types* such as (95).

(95) $(\textit{InfoState} \wedge \left[\text{ private } : \left[\text{ agenda } : \left[\text{ fst } : \textit{AssertionType}\ \right]\ \right]\ \right])$

In general $a : (T_1 \wedge T_2)$ just in case $a : T_1$ and $a : T_2$.

We can introduce meet (also known as intersection or conjunctive types) into a type system as in (96), repeated in Appendix 8.

(96) A system of complex types **TYPE**$_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has meet types* if

1. for any $T_1, T_2 \in \mathbf{Type}$, $(T_1 \wedge T_2) \in \mathbf{Type}$
2. for any $T_1, T_2 \in \mathbf{Type}$, $a :_{\mathbf{TYPE_C}} (T_1 \wedge T_2)$ iff $a :_{\mathbf{TYPE_C}} T_1$ and $a :_{\mathbf{TYPE_C}} T_2$

In our informal proof theoretic notation this can be represented as (97).

(97) For Γ a system of complex types

a. $$\frac{\Gamma \vdash T_1 \in \textbf{Type} \qquad \Gamma \vdash T_2 \in \textbf{Type}}{\Gamma \vdash (T_1 \wedge T_2) \in \textbf{Type}}$$

b. $$\frac{\Gamma \vdash a : T_1 \qquad \Gamma \vdash a : T_2}{\Gamma \vdash a : (T_1 \wedge T_2)}$$

c. $$\frac{\Gamma \vdash a : (T_1 \wedge T_2)}{\Gamma \vdash a : T_1}$$

d. $$\frac{\Gamma \vdash a : (T_1 \wedge T_2)}{\Gamma \vdash a : T_2}$$

Just as we did for join types we can introduce a generalized version of meet types as in (98), repeated in Appendix 8.

(98) A system of complex types $\mathbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has generalized meet types* if

1. for any non-empty finite set of types, $\mathbb{T}$, such that $\mathbb{T} \subseteq \mathbf{Type}$, $\bigwedge\mathbb{T} \in \mathbf{Type}$
2. for any finite $\mathbb{T} \subseteq \mathbf{Type}$, $a :_{\mathbf{TYPE_C}} \bigwedge\mathbb{T}$ iff $a :_{\mathbf{TYPE_C}} T$ for all $T \in \mathbb{T}$

In our informal proof theoretic notation this can be represented as (99).

(99) For Γ a system of complex types

a. $\dfrac{\Gamma \vdash T_1, \ldots, T_n \in \textbf{Type}}{\Gamma \vdash \bigwedge\{T_1, \ldots, T_n\} \in \textbf{Type}}$

b. $\dfrac{\Gamma \vdash a : T_1, \ldots, a : T_n}{\Gamma \vdash a : \bigwedge\{T_1, \ldots, T_n\}}$

c. $\dfrac{\Gamma \vdash a : \bigwedge\{T_1, \ldots, T_n\} \qquad 1 \leq i \leq n}{\Gamma \vdash a : T_i}$

As with join types, we can, if we wish, use $T_1 \wedge \ldots \wedge T_n$ to represent $\bigwedge\{T_1, \ldots, T_n\}$.

If T_1 and T_2 are record types then there will always be a record type (not a meet) T_3 which is necessarily equivalent to $T_1 \wedge T_2$, that is any record, r of type $T_1 \wedge T_2$ will be of type T_3 and *vice versa*. We will call this the *merge* of T_1 and T_2 which we will represent as $T_1 \dot{\wedge} T_2$ (with a dot under '$\wedge$'). For example, (100a) will have the same set of witnesses as (100b).

(100) a. $\left[\begin{array}{lll} \mathrm{f} & : & T_1 \end{array}\right] \wedge \left[\begin{array}{lll} \mathrm{g} & : & T_2 \end{array}\right]$

b. $\left[\begin{array}{lll} \mathrm{f} & : & T_1 \\ \mathrm{g} & : & T_2 \end{array}\right]$

When a label only occurs in one of the types being merged then the field with that label is also a field of the merge of the two types. Thus (101) holds.

(101) $\left[\begin{array}{lll} \text{f} & : & T_1 \end{array}\right] \dot{\wedge} \left[\begin{array}{lll} \text{g} & : & T_2 \end{array}\right] = \left[\begin{array}{lll} \text{f} & : & T_1 \\ \text{g} & : & T_2 \end{array}\right]$

When the same label, ℓ, occurs in both types, then whatever occurs in the ℓ-field must be of the types required in those fields by both the types. Thus (102a) and (102b) will have the same witnesses.

(102) a. $\left[\begin{array}{lll} \text{f} & : & T_1 \end{array}\right] \wedge \left[\begin{array}{lll} \text{f} & : & T_2 \end{array}\right]$

b. $\left[\begin{array}{lll} \text{f} & : & T_1 \wedge T_2 \end{array}\right]$

In a case like this we will make the merge recursive down inside the type, that is, we will make merge the types in the ‘f’-field in (102b). Thus (103) will hold.

(103) $\left[\begin{array}{lll} \text{f} & : & T_1 \end{array}\right] \dot{\wedge} \left[\begin{array}{lll} \text{f} & : & T_2 \end{array}\right] = \left[\begin{array}{lll} \text{f} & : & T_1 \dot{\wedge} T_2 \end{array}\right]$

If one or the other of T_1 and T_2 is not a record type then $T_1 \dot{\wedge} T_2$ will be $T_1 \wedge T_2$. If $T_1 \sqsubseteq T_2$ then $T_1 \dot{\wedge} T_2$ is T_1 and if $T_2 \sqsubseteq T_1$ then $T_1 \dot{\wedge} T_2$ is T_2.

We can define the merge operation on types in the following way (repeated in Appendix 12). The definition is closely related to the unification algorithms used in feature based grammar (see Shieber, 1986 for the classic reference).

We define a function μ which maps meets of record types to an equivalent record type, record types to equivalent types where meets in their values have been simplified by μ and any other types to themselves. μ is also defined on labelled sets which are not types in order to account for merging inside a record type of structures which depend on fields in the type outside the structure:

1. if for some $T_1, T_2, T = (T_1 \wedge T_2)$ and $T_1 \sqsubseteq T_2$ then $\mu(T) = T_1$
2. if for some $T_1, T_2, T = (T_1 \wedge T_2)$ and $T_2 \sqsubseteq T_1$ then $\mu(T) = T_2$
3. otherwise:
   a) if for some labelled sets $T_1, T_2, T = (T_1 \wedge T_2)$ then $\mu(T) = \mu'(\mu(T_1) \wedge \mu(T_2))$.
   b) if T is a labelled set then $\mu(T)$ is T' such that for any ℓ,v, $\langle \ell, \mu(v)\rangle \in T'$ iff $\langle \ell, v\rangle \in T$.
   c) otherwise $\mu(T) = T$.

$\mu'(T_1 \wedge T_2)$ is defined by:

1. if T_1 and T_2 are labelled sets, then $\mu'(T_1 \wedge T_2) = T_3$ such that
   a) for any ℓ, v_1, v_2, if $\langle \ell, v_1 \rangle \in T_1$ and $\langle \ell, v_2 \rangle \in T_2$, then
      i. if v_1 and v_2 are
         $\langle \lambda u_1{:}T'_1 \ldots \lambda u_i{:}T'_i \,.\, \phi, \langle \pi_1 \ldots \pi_i \rangle \rangle$
         and
         $\langle \lambda u'_1{:}T''_1 \ldots \lambda u'_k{:}T''_k \,.\, \psi, \langle \pi'_1 \ldots \pi'_k \rangle \rangle$
         respectively, then
         $\langle \lambda u_1{:}T'_1 \ldots \lambda u_i{:}T'_i, \lambda u'_1{:}T''_1 \ldots \lambda u'_k{:}T''_k \,.\, \mu(\phi \wedge \psi), \langle \pi_1 \ldots \pi_i, \pi'_1 \ldots \pi'_k \rangle \rangle \in T_3$
      ii. if v_1 is
         $\langle \lambda u_1{:}T'_1 \ldots \lambda u_i{:}T'_i \,.\, \phi, \langle \pi_1 \ldots \pi_i \rangle \rangle$
         and v_2 is a type (i.e. not of the form $\langle f, \Pi \rangle$ for some function f and sequence of paths Π), then
         $\langle \lambda u_1{:}T'_1 \ldots \lambda u_i{:}T'_i \,.\, \mu(\phi \wedge v_2), \langle \pi_1 \ldots \pi_i \rangle \rangle \in T_3$
      iii. if v_2 is
         $\langle \lambda u'_1{:}T''_1 \ldots \lambda u'_k{:}T''_k \,.\, \psi), \langle \pi'_1 \ldots \pi'_k \rangle \rangle$
         and v_1 is a type, then
         $\langle \lambda u'_1{:}T''_1 \ldots \lambda u'_k{:}T''_k \,.\, \mu(v_1 \wedge \psi)), \langle \pi'_1 \ldots \pi'_k \rangle \rangle \in T_3$
      iv. otherwise $\langle \ell, \mu(v_1 \wedge v_2) \rangle \in T_3$
   b) for any ℓ, v_1, if $\langle \ell, v_1 \rangle \in T_1$ and there is no v_2 such that $\langle \ell, v_2 \rangle \in T_2$, then $\langle \ell, v_1 \rangle \in T_3$
   c) for any ℓ, v_2, if $\langle \ell, v_2 \rangle \in T_2$ and there is no v_1 such that $\langle \ell, v_1 \rangle \in T_1$, then $\langle \ell, v_2 \rangle \in T_3$
2. if T_1 is $\text{list}(T'_1)$ ($\text{set}(T'_1)$, $\text{plurality}(T'_1)$) and T_2 is $\text{list}(T'_2)$ ($\text{set}(T'_2)$, $\text{plurality}(T'_2)$), then $\mu'(T_1 \wedge T_2) = \text{list}(\mu(T'_1 \wedge T'_2))$ ($\text{set}(\mu(T'_1 \wedge T'_2))$, $\text{plurality}(\mu(T'_1 \wedge T'_2))$)
3. otherwise $\mu'(T_1 \wedge T_2) = T_1 \wedge T_2$

$(T_1 \dot{\wedge} T_2)$ is used to represent $\mu(T_1 \wedge T_2)$. We call $(T_1 \dot{\wedge} T_2)$ the *merge* of T_1 and T_2.

It is important for this characterization of merge to work properly that paths in record types extend into meet types within the record type. Consider the merge expressed in (104a) where we assume that *Sit* is a basic type, for example the type of situations. According to the above definition (104a) will be identical with (104b).

(104) a. $$\left[\begin{array}{lll} \text{e} & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{y} & : & \textit{Ind} \\ \text{e} & : & \text{hug(x,y)} \end{array}\right] \\ \text{c}_1 & : & \text{boy(e.x)} \\ \text{c}_2 & : & \text{dog(e.y)} \end{array}\right] \underset{\cdot}{\wedge} \left[\begin{array}{lll} \text{e} & : & \textit{Sit} \end{array}\right]$$

b. $$\left[\begin{array}{lll} \text{e} & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{y} & : & \textit{Ind} \\ \text{e} & : & \text{hug(x,y)} \end{array}\right] \wedge \textit{Sit} \\ \text{c}_1 & : & \text{boy(e.x)} \\ \text{c}_2 & : & \text{dog(e.y)} \end{array}\right]$$

If the paths of record types do not extend into meet types then the paths of (104b) would be (105) and (104) would not be a well-formed record type since the ‘c_1’ and ‘c_2’-fields would reference non-existant paths.

(105) $\{\text{e}, \text{c}_1, \text{c}_2\}$

However, clearly any record which is of the type (104b) must in addition have the paths ‘e.x’ and ‘e.y’ because of the requirement expressed by the meet type. Thus it is appropriate and intuitive that these should count as paths in the record type thus making (104b) well-formed.

In our update action rules we will make use of an operation called *asymmetric merge*. The asymmetric merge of types T_1 and T_2, $T_1\boxed{\underset{\cdot}{\wedge}}T_2$ is like their merge except that if either T_1 or T_2 is not a record type then $T_1\boxed{\underset{\cdot}{\wedge}}T_2$ is T_2. Also asymmetric merge does not check for subtyping in the way that ordinary merge does. Thus when asymmetrically merging non-record types, T_1 and T_2, $T_1\boxed{\underset{\cdot}{\wedge}}T_2$ will always be T_2 regardless of whether the subtype relation holds between the two types.

We can define asymmetric merge in the following way (repeated in Appendix 12). The definition is closely related to the priority unification algorithms used in feature based grammar (Shieber, 1986).

The *asymmetric merge* of T_1 and T_2 is defined by a function, μ_{asym}, exactly like μ except that the first two clauses of the definition of μ are missing and μ' is replaced by another function μ'_{asym}. Thus the definition of μ_{asym} is:

1. if for some $T_1, T_2, T = (T_1 \wedge T_2)$ then $\mu_{\text{asym}}(T) = \mu'_{\text{asym}}(\mu_{\text{asym}}(T_1) \wedge \mu_{\text{asym}}(T_2))$.

2. if T is a record type then $\mu_{\text{asym}}(T)$ is T' such that for any ℓ,v, $\langle \ell, \mu_{\text{asym}}(v)\rangle \in T'$ iff $\langle \ell, v\rangle \in T$.
3. otherwise $\mu_{\text{asym}}(T) = T$.

The definition of μ'_{asym} is exactly like μ', replacing μ and μ' with μ_{asym} and μ'_{asym} respectively, except that the clause 3 of the definition of μ' is replaced by

3′. otherwise $\mu'_{\text{asym}}(T_1 \wedge T_2) = T_2$

We use $T_1 \boxed{\dot\wedge} T_2$ to represent the asymmetric merge of T_1 and T_2.

Asymmetric merge may result in an ill-formed record type if we take the asymmetric merge of a record type, T_1, and a non-record type, T_2, since T_1 may be embedded in a larger type with fields dependent on paths into T_1 which will not be present in the result where T_2 has been substituted for T_1 thus removing the relevant paths. Consider the asymmetric merge (106a) which is (106b), where, as above, *Sit* is a basic type.

(106) a. $\left[\begin{array}{lll} \text{e} & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{y} & : & \textit{Ind} \\ \text{e} & : & \text{hug(x,y)} \end{array}\right] \\ \text{c}_1 & : & \text{boy(e.x)} \\ \text{c}_2 & : & \text{dog(e.y)} \end{array}\right] \boxed{\dot\wedge} \left[\begin{array}{lll} \text{e} & : & \textit{Sit} \end{array}\right]$

b. $\left[\begin{array}{lll} \text{e} & : & \textit{Sit} \\ \text{c}_1 & : & \text{boy(e.x)} \\ \text{c}_2 & : & \text{dog(e.y)} \end{array}\right]$

(106b) is clearly not a well-formed type since the fields labelled by ‘c_1’ and ‘c_2’ address non-existent paths.

For the sake of saving space and for readability it will be useful to introduce the notational convention in (107) inspired by a similar notation in Head Driven Phrase Structure Grammar (Sag *et al.*, 2003).

(107) $\begin{bmatrix} T & & \\ \ell_1 & : & T_1 \\ \vdots & & \\ \ell_n & : & T_n \end{bmatrix}$

represents

$(T\dot{\wedge}\begin{bmatrix} \ell_1 & : & T_1 \\ \vdots & & \\ \ell_n & : & T_n \end{bmatrix})$

Armed with this technology we can define an update function, $\mathrm{f}_{\textsc{PlanAckAss}}$, and action rule which plan an acknowledgement to an assertion as in (108).

(108) a. $\mathrm{f}_{\textsc{PlanAckAss}}$

$\lambda r{:}\mathit{InfoState}$.

$\lambda u{:}\mathit{Assertion}$.

$$\begin{bmatrix} \text{private:} & \begin{bmatrix} \text{agenda:} & \begin{bmatrix} \text{fst:} & \begin{bmatrix} \text{s-event:} & \begin{bmatrix} \mathit{SEvent} \\ \text{sp}=u.\text{s-event.au}{:}\mathit{Ind} \\ \text{au}=u.\text{s-event.sp}{:}\mathit{Ind} \end{bmatrix} \\ \text{cont}=u.\text{cont}{:}\mathit{Cont} & \\ \text{illoc:acknowledge(s-event, cont)} & \end{bmatrix} \\ \text{rst}=r.\text{private.agenda:list}(\mathit{RecType}) & \end{bmatrix} \end{bmatrix} \\ \text{shared:} & \begin{bmatrix} \text{latest-utterance}=u{:}\mathit{Assertion} \end{bmatrix} \end{bmatrix}$$

b. PLANACKASS

$$\frac{s_{i,A} :_A T_{\text{curr}} \quad T_{\text{curr}} \sqsubseteq \text{domtype}(\mathrm{f}_{\textsc{PlanAckAss}}) \quad u^* :_A T_{\text{utt}} \quad T_{\text{utt}} \sqsubseteq \mathit{Assertion}}{s_{i+1,A} :_A T_{\text{curr}} \boxed{\dot{\wedge}} (\mathrm{f}_{\textsc{PlanAccAss}}(s_{i,A})(u^*) \dot{\wedge} \begin{bmatrix}\text{shared:} \begin{bmatrix}\text{latest-utterance:}T_{\text{utt}}\end{bmatrix}\end{bmatrix})}$$

(108a) maps information states (records), r, to a function that maps events to a type of information state. The second argument to the function (represented by u) requires a speech event which is an assertion. The type that results from applying the function to its arguments represents the effect of the update. This type requires the agenda to be result of pushing the type of an acknowledgement of the content of u onto the agenda and recording u as the latest utterance. It also requires that the speaker of the acknowledgement is the addressee of u and that the addressee of the acknolwedgement be the speaker of u. The content of the acknowledgement is the same as the content of the assertion. That is, what is being acknowledged is the content of the assertion.

(108b) we give the action rule PLANACKASS which characterizes the conditions under which an agent A can be licensed to plan to acknowledge an assertion. As before we use $s_{i,A}$ for A's current information state and $s_{i+1,A}$ for A's updated information state. T_{curr} is used for the type

A assigns to her current information state. u^* is used for the current utterance and T_{utt} for the type that A assigns to the current utterance. The rule says that if T_{curr} is a subtype of the domain type of the update function (108a) and T_{utt} is a subtype of *Assertion* then A is licensed to judge that her updated information state is of the type T_{curr} asymmetrically merged with the result of applying the update function to the current information state and the current utterance and merged with the information that the latest utterance is of type T_{utt}.

Note that the field 'shared.commitments' not been updated after the assertion. This is because the assertion has not yet been acknowledged. This models cases in which agents are *cautious* and do not assume that commitments are shared until the dialogue participant(s) they are addressing have confirmed acceptance. This interaction is known as grounding and is discussed (among other places) in Traum (1994) and Larsson (2002).

It is at the point that an agent performs an acknowledge-event ("ok") which will license an update of shared.commitments. Before we define this update function and action rule we will examine what needs to happen in order to update the commitments.

Suppose that in the dialogue so far it has been established that Dudamel is a conductor and that this is represented by the record type (109).

(109) $\left[\ \text{e} \quad : \quad \text{conductor(Dudamel)}\ \right]$

Suppose further that the latest utterance has the content that Beethoven is a composer, namely (110).

(110) $\left[\ \text{e} \quad : \quad \text{composer(Beethoven)}\ \right]$

One obvious way to combine them would be to merge them, that is, (111a) which is identical with (111b) which in turn is identical with (111c), given the definition in Appendix 12 which requires that the merge of any two types which are not both record types is identical with the meet of the two types.

(111) a. $\left[\ \text{e} \quad : \quad \text{conductor(Dudamel)}\ \right] \dot{\wedge} \left[\ \text{e} \quad : \quad \text{composer(Beethoven)}\ \right]$

b. $\left[\ \text{e} \quad : \quad \text{conductor(Dudamel)} \dot{\wedge} \text{composer(Beethoven)}\ \right]$

c. $\left[\ \text{e} \quad : \quad \text{conductor(Dudamel)} \wedge \text{composer(Beethoven)}\ \right]$

For the simple storing of information represented by predicates and names represented in (72) this might be sufficient. It makes the claim that all the information is collected into one eventuality. In more narrative dialogues referring to separate events which we may wish to be able to refer back to this would be an inadequate solution, however. It would be better if we have a way of keeping the labels ‘e’ separate so that they don’t clash, for example in (112a) which is identical with (112b)

(112) a. $\left[\begin{array}{lll} \mathrm{e}_1 & : & \text{conductor(Dudamel)} \end{array}\right] \dot{\wedge} \left[\begin{array}{lll} \mathrm{e}_2 & : & \text{composer(Beethoven)} \end{array}\right]$

b. $\left[\begin{array}{lll} \mathrm{e}_1 & : & \text{conductor(Dudamel)} \\ \mathrm{e}_2 & : & \text{composer(Beethoven)} \end{array}\right]$

The potential problems of label clash become very clear if we consider the types in (113a) corresponding to *a boy hugged a dog* and *a girl stroked a cat*. (113a) is identical with (113b) and has a single individual which is both a girl and a boy stroking another individual which is both a dog and a cat.

(113) a. $\left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_{\mathrm{boy}} & : & \mathrm{boy(x)} \\ \mathrm{y} & : & \textit{Ind} \\ \mathrm{c}_{\mathrm{dog}} & : & \mathrm{dog(y)} \\ \mathrm{e} & : & \mathrm{hug(x,y)} \end{array}\right] \dot{\wedge} \left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_{\mathrm{girl}} & : & \mathrm{girl(x)} \\ \mathrm{y} & : & \textit{Ind} \\ \mathrm{c}_{\mathrm{cat}} & : & \mathrm{cat(y)} \\ \mathrm{e} & : & \mathrm{stroke(x,y)} \end{array}\right]$

b. $\left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_{\mathrm{boy}} & : & \mathrm{boy(x)} \\ \mathrm{c}_{\mathrm{girl}} & : & \mathrm{girl(x)} \\ \mathrm{y} & : & \textit{Ind} \\ \mathrm{c}_{\mathrm{dog}} & : & \mathrm{dog(y)} \\ \mathrm{c}_{\mathrm{cat}} & : & \mathrm{cat(y)} \\ \mathrm{e} & : & \mathrm{hug(x,y)}\wedge\mathrm{stroke(x,y)} \end{array}\right]$

One way to get around this problem is to ensure that whenever you introduce new types you always use fresh labels that have not been used before and then use explicit constraints to require identity in cases where it is required. However, when we come to examine compositional semantics in Chapter 3 we will see that it is quite important to refer to particular labels in our rules of combination. Instead of introducing unique *labels* we will use the power of records to introduce unique *paths* when contents are combined. We will use the label ‘prev’ (“previous”). If T_{old} is the content so far and T_{new} is the content we wish to add then the new combined content will be as in (114a). Thus adding the content of *a girl stroked a cat* to that of *a boy hugged a dog* will yield (114b).

(114) a. $\left[\begin{array}{lll} \mathrm{prev} & : & T_{\mathrm{old}} \end{array}\right] \dot{\wedge}\, T_{\mathrm{new}}$

b. $$\left[\begin{array}{lll}\text{prev} & : & \left[\begin{array}{lll}\text{x} & : & \textit{Ind}\\ \text{c}_{\text{boy}} & : & \text{boy(x)}\\ \text{y} & : & \textit{Ind}\\ \text{c}_{\text{dog}} & : & \text{dog(y)}\\ \text{e} & : & \text{hug(x,y)}\end{array}\right]\\ \text{x} & : & \textit{Ind}\\ \text{c}_{\text{girl}} & : & \text{girl(x)}\\ \text{y} & : & \textit{Ind}\\ \text{c}_{\text{cat}} & : & \text{cat(y)}\\ \text{e} & : & \text{stroke(x,y)}\end{array}\right]$$

In the case of our example with Dudamel and Beethoven the result will be (115).

(115) $$\left[\begin{array}{lll}\text{prev} & : & \left[\begin{array}{lll}\text{e} & : & \text{conductor(Dudamel)}\end{array}\right]\\ \text{e} & : & \text{composer(Beethoven)}\end{array}\right]$$

If we add a further fact to this, say, that Uchida is a pianist we would obtain (116)

(116) $$\left[\begin{array}{lll}\text{prev} & : & \left[\begin{array}{lll}\text{prev} & : & \left[\begin{array}{lll}\text{e} & : & \text{conductor(Dudamel)}\end{array}\right]\\ \text{e} & : & \text{composer(Beethoven)}\end{array}\right]\\ \text{e} & : & \text{pianist(Uchida)}\end{array}\right]$$

This means that we now have to add additional information if we want to require identity, for example if we want the Beethoven and Uchida eventualities (prev.e and e in (116)) to be identical. We will return to these matters when we deal with anaphora in Chapter 3. Note that this strategy also gives us a straightforward record of the order in which content was added.

The update function $\text{f}_{\text{INTEGACK}}$ and action rule INTEGACK which allow for the integration of an acknowledgement into an information state are given in (117).

(117) a. $\text{f}_{\text{INTEGACK}}$

λr:*InfoState* .

λu:*Acknowledgement* .

$$\left[\text{shared:}\left[\begin{array}{l}\text{commitments=}\left[\text{prev:}r\text{.shared.commitments}\right] \mathbin{\dot{\wedge}} u\text{.cont:}\textit{RecType}\\ \text{latest-utterance=}u\text{:}\textit{Acknowledgement}\end{array}\right]\right]$$

b. INTEGACK

$$\frac{s_{i,a} :_A T_{\text{curr}} \quad T_{\text{curr}} \sqsubseteq \text{domtype}(\text{f}_{\text{IntegAck}}) \quad u^* :_A T_{\text{utt}} \quad T_{\text{utt}} \sqsubseteq \textit{Acknowledgement}}{s_{i+1,A} :_A T_{\text{curr}} \boxed{\dot{\wedge}} (\text{f}_{\text{INTEGACK}}(s_{i,A})(u^*) \mathbin{\dot{\wedge}} \left[\text{shared:}\left[\text{latest-utterance:}T_{\text{utt}}\right]\right])}$$

The update function (117a) takes an information state and an acknowledgement to a type of information state where the content of the acknowledgement is used to update the shared commitments and the acknowledgement utterance is recorded as the latest utterance. The action rule (117b) is parallel to PLANACKASS.

We need two more action rules relating to the agenda: EXECTOPAGENDA which allows the agent to make their contribution to executing what is uppermost on the agenda and DOWNDATEAGENDA which allows the removal of a type from the top of the agenda if an the current event is of that type. These two rules are given in (118).

(118) a. EXECTOPAGENDA

$$\frac{s_{i,A} :_A \begin{bmatrix} \textit{InfoState} \\ \text{private:} \begin{bmatrix} \text{agenda:} \begin{bmatrix} \text{fst:}\textit{RecType} \\ \text{rst:list}(\textit{RecType}) \end{bmatrix} \end{bmatrix} \end{bmatrix}}{:_A s_{i,A}.\text{private.agenda.fst!}}$$

b. DOWNDATEAGENDA

$$\frac{s_{i,A} :_A T_{\text{curr}} \qquad T_{\text{curr}} \sqsubseteq \begin{bmatrix} \text{private:} \begin{bmatrix} \text{agenda:} \begin{bmatrix} \text{fst:}\textit{RecType} \\ \text{rst:list}(\textit{RecType}) \end{bmatrix} \end{bmatrix} \end{bmatrix} \qquad u^* :_A s_{i,A}.\text{private.agenda.fst}}{s_{i+1,A} :_A T_{\text{curr}} \boxed{\dot{\wedge}} \begin{bmatrix} \text{private:} \begin{bmatrix} \text{agenda=}s_{i,A}.\text{private.agenda.rst:list}(\textit{RecType}) \end{bmatrix} \end{bmatrix}}$$

2.7 English resources

We shall say that update functions and action rules are different kinds of *resources* that are available to an agent. Those that we have discussed in the previous section are resources associated with *dialogue management*. We shall see more examples of resources as the book progresses. In general they can be viewed as related to the theory of topoi and enthymemes as discussed in Breitholtz' work (Breitholtz and Villing, 2008; Breitholtz, 2010; Breitholtz and Cooper, 2011; Breitholtz, 2014b, fthc).

In this chapter we are taking signs to be objects of type (69) and the sign corresponding to *Dudamel is a conductor* is (73). For compactness of representation we can define an operation which takes a speech event type and a content and constructs the corresponding sign. This can be defined as in (119).

(119) If σ is a type of speech event and κ is a type (of situation) then

$$\text{sign}(\sigma,\kappa) = \begin{bmatrix} \text{s-event:} \begin{bmatrix} \text{e:}\sigma \end{bmatrix} \\ \text{cont=} \begin{bmatrix} \text{e:}\kappa \\ \text{c}_{\text{tns}}\text{:final\_align}(\Uparrow\text{s-event.e,e}) \end{bmatrix} \text{:}\textit{RecType} \end{bmatrix}$$

Note that the operation ‘sign’ introduces the interpretation of present tense (represented by the field ‘$\mathrm{c}_{\mathrm{tns}}$’). This is only possible because the resources we are considering concern only simple present tense assertions such as *Dudamel is a conductor*. We will see already in the next chapter that things are not this simple. We can use (119) to create signs types for utterances with specific contents such as *Dudamel is a conductor* or *Beethoven is a composer*. We will use another operation ‘sign_{uc}’ to create signs with underspecified content as defined in (120).

(120) If σ is a type of speech event then

$$\mathrm{sign}_{uc}(\sigma) = \begin{bmatrix} \text{s-event:} \begin{bmatrix} \text{e:}\sigma \end{bmatrix} \\ \text{cont:}\textit{RecType} \end{bmatrix}$$

Now we can characterize the sign types that an agent that can deal with the simple dialogues that we have been characterizing in this chapter as (121).

(121) {sign(“Dudamel is a conductor”, conductor(dudamel)),
sign(“Beethoven is a composer”, composer(beethoven)),
sign(“Uchida is a pianist”, pianist(uchida)),
sign_{uc}(“ok”),
sign_{uc}(“aha”)}

Recall that “Dudamel is a conductor” etc. represent a type of a string of word utterance events. For any word w, “w” is the type of event where w is uttered. For present purposes we assume that the agent has basic types of word utterances as given in (122a). In order to cope with the content the agent must have a basic type *Ind* to which certain individuals belong as given in (122b). Finally in order to construct the ptypes used for the content the agent would have to have the predicates given in (122c).

(122) a. “Dudamel”, “is”, “a”, “conductor”, “Beethoven”, “composer”, “Uchida”, “pianist”, “aha”, “ok”

b. dudamel, beethoven, uchida : *Ind*

c. predicates with arity ⟨*Ind*⟩: conductor, composer, pianist

The set of ptypes based on (122b,c) is thus (123).

(123) $\{p(a) \mid p \in \{\mathrm{conductor}, \mathrm{composer}, \mathrm{pianist}\}$ and $a \in \{\mathrm{dudamel}, \mathrm{beethoven}, \mathrm{uchida}\}\}$

Of the ptypes in (123) we could say that 'conductor(dudamel)', 'composer(beethoven)' and 'pianist(uchida)' are non-empty ("true") and the rest are empty, although that may not correspond to the actual facts of the world. (Beethoven was a pianist, for example.) Very often, we are mainly interested in whether a ptype has witnesses (something of the type) or not and not particularly what those witnesses are. In a complete formal treatment, of course, the type system would specify objects which belong to those types. For example, we could say s_1 : conductor(dudamel), s_2 : composer(beethoven) and s_3 : pianist(uchida). Informally, we can say s_1 is a situation where Dudamel is a conductor or which shows that Dudamel is a conductor and so on. The idea of saying that an agent has a certain type in its resources is not so much to say that it has complete information about what belongs to the type (although its memory will contain partial information about what belongs to what types) but rather that it has a way (possibly not entirely decidable) of recognizing an object of the type if it sees one. Thus since I am an agent with the type 'composer(uchida)' in my resources I know (sort of) what it would mean for a situation to be of this type, e.g. a situation in which Uchida has written original musical compositions, had them performed and so on. When we are using our type theory to give an analysis of certain fragments of language we are sometimes interested in going into more detail concerning the criteria for belonging to a given type. Other times we just treat the type as basic and only need to assume that the agent has some way of recognizing objects of the type. It depends on the level of detail we are interested in for the particular analysis.

Let us now check that we can characterize the types of information states of A and B in the dialogue (124), where we represent the information states associated with the two agents at various points in the dialogue as a_i and b_i, and the utterance events as u_i.

(124)

	a_0,b_0	
A:	Dudamel is a conductor	u_1
	a_1,b_1	
B:	Aha	u_2
	a_2,b_2	
A:	Beethoven is a composer	u_3
	a_3,b_3	
B:	ok	u_4
	a_4,b_4	
A:	Uchida is a pianist	u_5
	a_5,b_5	
B:	ok	u_6
	a_6,b_6	

We will assume that a_0 and b_0 are initial states, essentially empty except for A's agenda to make the three assertions. This is shown in (125).

(125) a. a_0 :

$$\left[\begin{array}{ll} \text{private:} & \left[\begin{array}{ll} \text{agenda:} & \left[\begin{array}{ll} \text{fst=} & \left[\begin{array}{l} \textit{Assertion} \\ \text{s-event:}\left[\text{sp=}A\text{:}\textit{Ind}\right] \\ \text{cont=}\left[\text{e:conductor(dudamel)}\right]\text{:}\textit{RecType} \end{array}\right]\text{:}\textit{RecType} \\ \text{rst:} & \left[\begin{array}{ll} \text{fst=} & \left[\begin{array}{l} \textit{Assertion} \\ \text{s-event:}\left[\text{sp=}A\text{:}\textit{Ind}\right] \\ \text{cont=}\left[\text{e:composer(beethoven)}\right]\text{:}\textit{RecType} \end{array}\right]\text{:}\textit{RecType} \\ \text{rst:} & \left[\begin{array}{ll} \text{fst=} & \left[\begin{array}{l} \textit{Assertion} \\ \text{s-event:}\left[\text{sp=}A\text{:}\textit{Ind}\right] \\ \text{cont=}\left[\text{e:pianist(uchida)}\right]\text{:}\textit{RecType} \end{array}\right]\text{:}\textit{RecType} \\ \multicolumn{2}{l}{\text{rst=[ ]:}\mathrm{list}(\textit{RecType})} \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right] \\ \text{shared:} & \left[\begin{array}{l} \text{latest-utterance:}\textit{ERec} \\ \text{commitments=}\textit{Rec}\text{:}\textit{RecType} \end{array}\right] \end{array}\right]$$

b. b_0 :

$$\left[\begin{array}{ll} \text{private:} & \left[\text{agenda=[ ]:}\mathrm{list}(\textit{RecType})\right] \\ \text{shared:} & \left[\begin{array}{l} \text{latest-utterance:}\textit{ERec} \\ \text{commitments=}\textit{Rec}\text{:}\textit{RecType} \end{array}\right] \end{array}\right]$$

(125) indicates that a_0 meets the premise of the action rule EXECTOPAGENDA and in accordance with this A creates the utterance u_1 which is of the first type on the agenda in a_0. The existence of u_1 will now trigger the action rule DOWNDATEAGENDA which will remove the first type on the agenda in a_0. From this we derive that there is an intermediate information state for A which is of the type (126).

(126)

$$\left[\begin{array}{ll} \text{private:} & \left[\begin{array}{ll} \text{agenda:} & \left[\begin{array}{ll} \text{fst=} & \left[\begin{array}{l} \textit{Assertion} \\ \text{s-event:}\left[\text{sp=}A\text{:}\textit{Ind}\right] \\ \text{cont=}\left[\text{e:composer(beethoven)}\right]\text{:}\textit{RecType} \end{array}\right]\text{:}\textit{RecType} \\ \text{rst:} & \left[\begin{array}{ll} \text{fst=} & \left[\begin{array}{l} \textit{Assertion} \\ \text{s-event:}\left[\text{sp=}A\text{:}\textit{Ind}\right] \\ \text{cont=}\left[\text{e:pianist(uchida)}\right]\text{:}\textit{RecType} \end{array}\right]\text{:}\textit{RecType} \\ \multicolumn{2}{l}{\text{rst=[ ]:}\mathrm{list}(\textit{RecType})} \end{array}\right] \end{array}\right] \end{array}\right] \\ \text{shared:} & \left[\begin{array}{l} \text{latest-utterance:}\textit{ERec} \\ \text{commitments=}\textit{Rec}\text{:}\textit{RecType} \end{array}\right] \end{array}\right]$$

This in turn triggers the action rule PLANACKASS which will put the type of a speech act on the top of the agenda which acknowledges u_0 with speaker B and addressee A and also record u_0 as the latest utterance. Thus the state a_1, A's state after u_0, will be of the type (127).

$$(127)\quad \left[\begin{array}{ll}
\text{private:} & \left[\begin{array}{ll}\text{agenda:} & \left[\begin{array}{ll}
\text{fst=} & \left[\begin{array}{l}\mathit{Acknowledgement}\\ \text{s-event:}\left[\begin{array}{l}\text{sp=}B\text{:}\mathit{Ind}\\ \text{au=}A\text{:}\mathit{Ind}\end{array}\right]\\ \text{cont=}u_1\text{.cont:}\mathit{RecType}\end{array}\right]\text{:}\mathit{RecType}\\
\text{rst:} & \left[\begin{array}{ll}
\text{fst=} & \left[\begin{array}{l}\mathit{Assertion}\\ \text{s-event:}\left[\text{sp=}A\text{:}\mathit{Ind}\right]\\ \text{cont=}\left[\text{e:composer(beethoven)}\right]\text{:}\mathit{RecType}\end{array}\right]\text{:}\mathit{RecType}\\
\text{rst:} & \left[\begin{array}{ll}
\text{fst=} & \left[\begin{array}{l}\mathit{Assertion}\\ \text{s-event:}\left[\text{sp=}A\text{:}\mathit{Ind}\right]\\ \text{cont=}\left[\text{e:pianist(uchida)}\right]\text{:}\mathit{RecType}\end{array}\right]\text{:}\mathit{RecType}\\
\multicolumn{2}{l}{\text{rst=[ ]:list}(\mathit{RecType})}
\end{array}\right]
\end{array}\right]
\end{array}\right]\end{array}\right]\\
\text{shared:} & \left[\begin{array}{ll}
\text{latest-utterance=}u_1\text{:} & \left[\begin{array}{l}\mathit{Assertion}\\ \text{s-event:}\left[\begin{array}{l}\text{sp=}A\text{:}\mathit{Ind}\\ \text{au=}B\text{:}\mathit{Ind}\end{array}\right]\\ \text{cont=}\left[\text{e:conductor(dudamel)}\right]\text{:}\mathit{RecType}\end{array}\right])\\
\multicolumn{2}{l}{\text{commitments=}\mathit{Rec}\text{:}\mathit{RecType}}
\end{array}\right]
\end{array}\right]$$

This assumes that A has judged u_1 to be of the type in (128) (corresponding to T_{utt} in PLANACK-ASS), as represented in the 'shared.latest-utterance'-field in (127).

$$(128)\quad \left[\begin{array}{l}\mathit{Assertion}\\ \text{s-event:}\left[\begin{array}{l}\text{sp=}A\text{:}\mathit{Ind}\\ \text{au=}B\text{:}\mathit{Ind}\end{array}\right]\\ \text{cont=}\left[\text{e:conductor(dudamel)}\right]\text{:}\mathit{RecType}\end{array}\right]$$

Thus the first type on A's agenda is now the type of an acknowledgement event in which B is the speaker and A is the audience. This means that A has to wait for B to make the acknowledgement and to pay attention to it before continuing with the next item on the agenda.

A type for b_1 can be obtained in a similar fashion using PLANACKASS, assuming that B also classifies u_1 as a witness of the type (128). This type is given in (129).

(129)
$$\left[\begin{array}{ll}
\text{private:} & \left[\begin{array}{ll}\text{agenda:} & \left[\begin{array}{l}
\text{fst}=\left[\begin{array}{ll}
\multicolumn{2}{l}{\textit{Acknowledgement}}\\
\text{s-event:} & \left[\begin{array}{l}\text{sp}=B{:}\textit{Ind}\\ \text{au}=A{:}\textit{Ind}\end{array}\right]\\
\multicolumn{2}{l}{\text{cont}=u_1.\text{cont}{:}\textit{RecType}}
\end{array}\right]{:}\textit{RecType}\\
\text{rst}=[\ ]{:}\text{list}(\textit{RecType})
\end{array}\right]\end{array}\right]\\
\text{shared:} & \left[\begin{array}{ll}
\text{latest-utterance}=u_1{:} & \left[\begin{array}{ll}
\multicolumn{2}{l}{\textit{Assertion}}\\
\text{s-event:} & \left[\begin{array}{l}\text{sp}=A{:}\textit{Ind}\\ \text{au}=B{:}\textit{Ind}\end{array}\right]\\
\multicolumn{2}{l}{\text{cont}=\left[\text{e:conductor(dudamel)}\right]{:}\textit{RecType}}
\end{array}\right]\\
\multicolumn{2}{l}{\text{commitments}=\textit{Rec}{:}\textit{RecType}}
\end{array}\right]
\end{array}\right]$$

Now we are in a situation where both A and B are in information states (a_1 and b_1) with non-empty agendas. A and B are *coordinated* because the type labelled by 'shared' on their respective gameboards are the same. In addition, they are coordinated because they both have the same type at the top of their respective agendas. That is, they are both plannning that the next move will be one in which B acknowledges the content of u_1.

EXECTOPAGENDA is applicable to both a_1 and b_1 followed by DOWNDATEAGENDA. In A's case realizing the acknowledgement type involves waiting for B to speak and paying attention to the acknowledgement when it comes. In B's case realizing the type involves making the utterance u_2. This is an elementary form of what is known as *turn-taking* in the dialogue literature (Sacks *et al.*, 1974).

Once u_2 has been uttered, its content has to be integrated into 'shared.commitments' on both A's and B's gameboards. This can be achieved by applying INTEGACK. The results for both gameboards are given in (130).

(130) a. a_2 :
$$\left[\begin{array}{ll}
\text{private:} & \left[\begin{array}{ll}\text{agenda:} & \left[\begin{array}{ll}
\text{fst}= & \left[\begin{array}{l}
\textit{Assertion}\\
\text{s-event:}\left[\text{sp}=A{:}\textit{Ind}\right]\\
\text{cont}=\left[\text{e:composer(beethoven)}\right]{:}\textit{RecType}
\end{array}\right]{:}\textit{RecType}\\
\text{rst:} & \left[\begin{array}{l}
\text{fst}=\left[\begin{array}{l}
\textit{Assertion}\\
\text{s-event:}\left[\text{sp}=A{:}\textit{Ind}\right]\\
\text{cont}=\left[\text{e:pianist(uchida)}\right]{:}\textit{RecType}
\end{array}\right]{:}\textit{RecType}\\
\text{rst}=[\ ]{:}\text{list}(\textit{RecType})
\end{array}\right]
\end{array}\right]\end{array}\right]\\
\text{shared:} & \left[\begin{array}{ll}
\text{latest-utterance}=u_2{:} & \left[\begin{array}{ll}
\multicolumn{2}{l}{\textit{Acknowledgement}}\\
\text{s-event:} & \left[\begin{array}{l}\text{sp}=B{:}\textit{Ind}\\ \text{au}=A{:}\textit{Ind}\end{array}\right]\\
\multicolumn{2}{l}{\text{cont}=\left[\text{e:conductor(dudamel)}\right]{:}\textit{RecType}}
\end{array}\right]\\
\text{commitments}= & \left[\begin{array}{l}\text{prev:}\textit{Rec}\\ \text{e:conductor(dudamel)}\end{array}\right]{:}\textit{RecType}
\end{array}\right]
\end{array}\right]$$

b. $$b_2 : \begin{bmatrix} \text{private:} & \begin{bmatrix}\text{agenda=[ ]:list}(\textit{RecType})\end{bmatrix} \\ \text{shared:} & \begin{bmatrix} \text{latest-utterance=}u_2\text{:} & \begin{bmatrix} \textit{Acknowledgement} \\ \text{s-event:} \begin{bmatrix}\text{sp=}B\text{:}\textit{Ind} \\ \text{au=}A\text{:}\textit{Ind}\end{bmatrix} \\ \text{cont=}\begin{bmatrix}\text{e:conductor(dudamel)}\end{bmatrix}\text{:}\textit{RecType}\end{bmatrix} \\ \text{commitments=} & \begin{bmatrix}\text{prev:}\textit{Rec}\\ \text{e:conductor(dudamel)}\end{bmatrix}\text{:}\textit{RecType} \end{bmatrix}\end{bmatrix}$$

A and B are coordinated in that they both hypothesize the same type for shared.commitments. Now the assertion-acknowledgement cycle can begin again and repeat until both agents have gameboards with empty agendas. The final gameboards for A and B are both of the type given in (131). Thus A and B are aligned at the end of this dialogue.

(131) $$\begin{bmatrix} \text{private:} & \begin{bmatrix}\text{agenda=[ ]:list}(\textit{RecType})\end{bmatrix} \\ \text{shared:} & \begin{bmatrix} \text{latest-utterance=}u_6\text{:} & \begin{bmatrix} \textit{Acknowledgement} \\ \text{s-event:} \begin{bmatrix}\text{sp=}B\text{:}\textit{Ind} \\ \text{au=}A\text{:}\textit{Ind}\end{bmatrix} \\ \text{cont=}\begin{bmatrix}\text{e:pianist(uchida)}\end{bmatrix}\text{:}\textit{RecType}\end{bmatrix} \\ \text{commitments=} & \begin{bmatrix} \text{prev:} \begin{bmatrix} \text{prev:} \begin{bmatrix}\text{prev:}\textit{Rec} \\ \text{e:conductor(dudamel)}\end{bmatrix} \\ \text{e:composer(beethoven)}\end{bmatrix} \\ \text{e:pianist(uchida)}\end{bmatrix}\text{:}\textit{RecType} \end{bmatrix}\end{bmatrix}$$

The dialogue we have analyzed here is, of course, extremely simple and we have assumed ideal conditions under which the dialogue participants completely understand each other. However, it seems that the tools developed in this chapter could be developed further to account for the kind of dialogue data, including misunderstandings and phenomena such as repair. In particular we could build on the kind of analyses presented in Ginzburg (2012). In the remainder of the book we will not develop this further but concentrate on what is involved in incorporating treatments of more traditional grammatical and semantic concerns into an action based framework capable of dealing with dialogue phenomena and examining what might be gained by taking a dialogical approach to these concerns.

2.8 Summary of resources introduced

2.8.1 Universal grammar resources

2.8.1.1 Types

Loc — a basic type

l : *Loc* iff l is a region in three dimensional space

Phon — a basic type

e : *Phon* iff e is a phonological event

$$\textit{SEvent} \text{ — } \begin{bmatrix} \text{e-loc} & : & \textit{Loc} \\ \text{sp} & : & \textit{Ind} \\ \text{au} & : & \textit{Ind} \\ \text{e} & : & \textit{Phon} \\ \text{c}_{\text{loc}} & : & \text{loc(e,e-loc)} \\ \text{c}_{\text{sp}} & : & \text{speaker(e,sp)} \\ \text{c}_{\text{au}} & : & \text{audience(e,au)} \end{bmatrix}$$

Cont — *RecType*

$$\textit{Sign} \text{ — } \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{cont} & : & \textit{Cont} \end{bmatrix}$$

SignType — a basic type

T : *SignType* iff $T \sqsubseteq$ *Sign*

2.8.1.2 Predicates

with arity $\langle$*Phon*, *Loc*$\rangle$

loc — e : loc(u, l) iff u is located at l in e

with arity $\langle$*Phon*, *Ind*$\rangle$

speaker — e : speaker(u, a) iff u is the speaker of u in e

audience — e : audience(u, a) iff u is the audience of u in e

2.8.2 Universal speech act resources

2.8.2.1 Types

$$\textit{Assertion} \text{ — } \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{cont} & : & \textit{RecType} \\ \text{illoc} & : & \text{assert(s-event, cont)} \end{bmatrix}$$

$$\textit{Query} \text{ — } \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{cont} & : & \textit{Question} \\ \text{illoc} & : & \text{query(s-event, cont)} \end{bmatrix}$$

$$\textit{Command} \;—\; \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{cont} & : & \textit{RecType} \\ \text{illoc} & : & \text{command(s-event, cont)} \end{bmatrix}$$

$$\textit{Acknowledgement} \;—\; \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{cont} & : & \textit{RecType} \\ \text{illoc} & : & \text{acknowledge(s-event, cont)} \end{bmatrix}$$

AssertionType — a basic type

T : *AssertionType* iff $T \sqsubseteq$ *Assertion*

QueryType — a basic type

T : *QueryType* iff $T \sqsubseteq$ *Query*

CommandType — a basic type

T : *CommandType* iff $T \sqsubseteq$ *Command*

AcknowledgementType — a basic type

T : *AcknowledgementType* iff $T \sqsubseteq$ *Acknowledgement*

2.8.3 Universal dialogue resources

2.8.3.1 Types

$$\textit{InfoState} \;—\; \begin{bmatrix} \text{private} & : & \begin{bmatrix} \text{agenda} & : & \text{list}(\textit{RecType}) \end{bmatrix} \\ \text{shared} & : & \begin{bmatrix} \text{latest-utterance} & : & \textit{Sign}^* \\ \text{commitments} & : & \textit{RecType} \end{bmatrix} \end{bmatrix}$$

$$\textit{InitInfoState} \;—\; \begin{bmatrix} \text{private} & : & \begin{bmatrix} \text{agenda=[ ]} & : & \text{list}(\textit{RecType}) \end{bmatrix} \\ \text{shared} & : & \begin{bmatrix} \text{latest-utterance=}\varepsilon & : & \textit{Sign}^* \\ \text{commitments=}\textit{Rec} & : & \textit{RecType} \end{bmatrix} \end{bmatrix}$$

2.8.3.2 Update functions and action rules

$$\text{f}_{\text{PLANACKASS}}\ \lambda r{:}\textit{InfoState}\ .\ \lambda u{:}\textit{Assertion}\ .\ \begin{bmatrix} \text{private:} & \begin{bmatrix} \text{agenda:} & \begin{bmatrix} \text{fst:} & \begin{bmatrix} \text{s-event:} & \begin{bmatrix} \textit{SEvent} \\ \text{sp=}u\text{.s-event.au:}\textit{Ind} \\ \text{au=}u\text{.s-event.sp:}\textit{Ind} \end{bmatrix} \\ \text{cont=}u\text{.cont:}\textit{Cont} & \\ \text{illoc:acknowledge(s-event, cont)} & \end{bmatrix} \\ \text{rst=}r\text{.private.agenda:list}(\textit{RecType}) & \end{bmatrix} \end{bmatrix} \\ \text{shared:} & \begin{bmatrix} \text{latest-utterance=}u\text{:}\textit{Assertion} \end{bmatrix} \end{bmatrix}$$

$$\text{PLANACKASS}\quad \frac{s_{i,A} :_A T_{\text{curr}} \qquad T_{\text{curr}} \sqsubseteq \text{domtype}(\text{f}_{\text{PLANACKASS}}) \qquad u^* :_A T_{\text{utt}} \qquad T_{\text{utt}} \sqsubseteq \textit{Assertion}}{s_{i+1,A} :_A T_{\text{curr}}\boxed{\dot{\wedge}}(\text{f}_{\text{PLANACCASS}}(s_{i,A})(u^*) \dot{\wedge} \left[\text{shared:}\left[\text{latest-utterance:}T_{\text{utt}}\right]\right])}$$

$$\text{f}_{\text{INTEGACK}}\quad \lambda r{:}\textit{InfoState}\ .\ \lambda u{:}\textit{Acknowledgement}\ .\ \left[\text{shared:}\begin{bmatrix}\text{commitments=}\left[\text{prev:}r\text{.shared.commitments}\right]\dot{\wedge}u\text{.cont:}\textit{RecType}\\ \text{latest-utterance=}u{:}\textit{Acknowledgement}\end{bmatrix}\right]$$

$$\text{INTEGACK}\quad \frac{s_{i,a} :_A T_{\text{curr}} \qquad T_{\text{curr}} \sqsubseteq \text{domtype}(\text{f}_{\text{IntegAck}}) \qquad u^* :_A T_{\text{utt}} \qquad T_{\text{utt}} \sqsubseteq \textit{Acknowledgement}}{s_{i+1,A} :_A T_{\text{curr}}\boxed{\dot{\wedge}}(\text{f}_{\text{INTEGACK}}(s_{i,A})(u^*)\ \dot{\wedge}\ \left[\text{shared:}\left[\text{latest-utterance:}T_{\text{utt}}\right]\right])}$$

$$\text{EXECTOPAGENDA}\quad \frac{s_{i,A} :_A \begin{bmatrix}\textit{InfoState}\\ \text{private:}\left[\text{agenda:}\begin{bmatrix}\text{fst:}\textit{RecType}\\ \text{rst:list}(\textit{RecType})\end{bmatrix}\right]\end{bmatrix}}{:_A\ s_{i,A}\text{.private.agenda.fst!}}$$

DOWNDATEAGENDA

$$\frac{s_{i,A} :_A T_{\text{curr}} \qquad T_{\text{curr}} \sqsubseteq \left[\text{private:}\left[\text{agenda:}\begin{bmatrix}\text{fst:}\textit{RecType}\\ \text{rst:list}(\textit{RecType})\end{bmatrix}\right]\right] \qquad u^* :_A s_{i,A}\text{.private.agenda.fst}}{s_{i+1,A} :_A T_{\text{curr}}\boxed{\dot{\wedge}}\ \left[\text{private:}\left[\text{agenda=}s_{i,A}\text{.private.agenda.rst:list}(\textit{RecType})\right]\right]}$$

2.8.4 English resources

2.8.4.1 Basic types and predicates

Basic phonological types for words

{"Dudamel", "is", "a", "conductor", "Beethoven", "composer", "Uchida", "pianist", "aha", "ok"}

Witnesses for basic types

Ind — dudamel, beethoven, uchida : *Ind*

Predicates

with arity ⟨*Ind*⟩ {conductor, composer, pianist}

2.8.4.2 Sign types

Notation

If σ is a type of speech event and κ is a type (of situation) then

$$\text{sign}(\sigma,\kappa)= \left[\begin{array}{l}\text{s-event:}\left[\text{e:}\sigma\right] \\ \text{cont=}\left[\begin{array}{l}\text{e:}\kappa \\ \text{c}_{\text{tns}}\text{:final\_align}(\Uparrow\text{s-event.e,e})\end{array}\right]\text{:}\mathit{RecType}\end{array}\right]$$

If σ is a type of speech event then

$$\text{sign}_{uc}(\sigma)= \left[\begin{array}{l}\text{s-event:}\left[\text{e:}\sigma\right] \\ \text{cont:}\mathit{RecType}\end{array}\right]$$

Lexical sign types

{sign("Dudamel is a conductor", conductor(dudamel)),
sign("Beethoven is a composer", composer(beethoven)),
sign("Uchida is a pianist", pianist(uchida)),
sign_{uc}("ok"),
sign_{uc}("aha")}

2.9 Summary

In this chapter we have been concerned with relating a general approach to event perception and action to the particular case of the kind of information states that are need to take part in linguistic events and how such events facilitate information exchange.

We started with the string theory of events based on important work by Tim Fernando. This is the idea that events can be seen as strings of smaller events and that our perception of events involves a string of observations which involve classifying smaller events as witnesses for certain types.

The second component in our approach is that the idea that there are various kinds of acts which can be associated with types. This builds on the notion of judgement which is present in Martin-Löf's type theory. We say that in addition to this type act one can also query whether objects belong to a type and, crucially for the proposals in this chapter, one can create witnesses (such as events) for a type.

Building on these notions we talked about non-linguistic games and the kind of information states which an agent must be in in order to coordinate with an agent in playing a game, that is, in creating an event of a type corresponding to the game. A crucial component of these information states is the agenda which indicates the types of events which the agent is currently planning to create witnesses for.

Associated with information states are update functions which will map from an information state of a certain type to the type of the next information state. Such update functions provide affordances which license the agent to progress from one information state to another. These affordances are treated in terms of action rules which look very much like inference rules in a logic except that they represent affordances rather than inferences.

We showed how this view of action in general can be applied to the special case of action in linguistic communication. In approaching linguistic action from this direction our aim is to move towards a theory of language related, for example, to that proposed by Christiansen and Chater (2008) where we ask not how the brain is adapted to language but rather how language is adapted to the brain. That is, we want to explore the extent to which language is based on prelinguistic cognitive abilities.

We characterized a type of speech events (although we pointed out that the type we used is not the only type for speech events, as shown, among other things, by split utterances). We developed a notion of sign and sign type closely related to the kind of signs which are found in the feature based theory HPSG. We showed how signs can be related to action rules corresponding to speech event understanding and speech event generation.

Finally, we proposed some simple update functions and action rules for dialogue which make use of the notion of sign and showed how these can be used to account for a very simple dialogue.

An important notion that we introduced is that of a cognitive resource available to an agent, a notion that we make precise in terms of types.

Chapter 3

Grammar in a theory of action

3.1 Introduction

In Chapter 2 we made the simplifying assumption that sentences come as single unanalyzed units (something like the assumption that is made in propositional logic). In this chapter we will deal with the same simple examples but break the sentences down into their constituent parts. (This will be something like moving from propositional logic to predicate logic without quantifiers.) In order to do this we will need more complex signs.

3.2 Constituent structure and events

We will first consider how linguistic constituent structure is related to our general perception of events. We have so far talked of events in terms of string types which we have related to finite state automata. Finite state automata are equivalent to regular grammars. We will now consider an example of how we perceive events which suggest a more complex structure in terms of strings of regular types. This gives us something which is equivalent to recursive transition networks (RTNs) which are in turn equivalent to context free grammars.[1] Consider an event type of bus trips, *BusTrip*. This could be defined as in (1).

(1) $\mathit{GetBus}^\frown\mathit{TravelOnBus}^\frown\mathit{GetOffBus}$

Each of the three event types which are concatenated in (1) could be further broken down into strings of events. For example, *GetBus* might be defined as in (2).

(2) $\mathit{WaitAtBusstop}^{*\frown}\mathit{BusArrive}^\frown\mathit{GetOnBus}$

[1] For a general introduction to automata theory and its relation to the Chomsky hierarchy see, for example, Partee *et al.* (1990).

The elements in (2) could be broken down further. For example, getting on the bus could be analyzed in terms of going towards a door on the bus, waiting for the door to open, placing one foot on the step into the bus and then the other, paying for your ticket and so on. There seems almost no limit to how finegrained an analysis of events we can give. Which muscles do you have to move in order to place your right foot inside the bus? What events are involved in the contraction of this muscle? However, there seems to be a limit on the level of detail we need to be conscious of (or even are capable of being conscious of) in order to carry out a high level action like getting on a bus. We can also build upwards from the type *BusTrip*. For example, many bus trips are not direct in that we have to change buses in order to reach our destination. Thus a bus trip can consist of a string of events where you get on a bus, travel on it and then get off it again. A return bus trip involves a bus trip from one place to another followed (after intervening events) by a bus trip from the second place back to the first. Both of those bus trips might involve several buses if the connection is not direct.

Given the two definitions in (1) and (2), *BusTrip* could be seen as an abbreviation for the regular string type in (3).

(3) $\mathit{WaitAtBusstop}^{*\frown}\mathit{BusArrive}^{\frown}\mathit{GetOnBus}^{\frown}\mathit{TravelOnBus}^{\frown}\mathit{GetOffBus}$

Thus while our notation is giving us the beginnings of a hierarchical organization, the type that is represented by the notation is not hierarchically organized. We are still in the realm of a finite state system. Compare this with the statements in (4).

(4) a. e : *BusTrip* iff e : $\mathit{GetBus}^{\frown}\mathit{TravelOnBus}^{\frown}\mathit{GetOffBus}$

b. e : *GetBus* iff e : $\mathit{WaitAtBusstop}^{*\frown}\mathit{BusArrive}^{\frown}\mathit{GetOnBus}$

The statements in (4) claim that there are distinct types *BusTrip* and *GetBus* in addition to the regular types used on the right-hand side of 'iff'. These types are *equivalent* to the regular types in the sense that anything of the one type will be of the other type. Now the type system is hierarchically organized and includes two additional "higher" types *BusTrip* and *GetBus*. On the face of it one might think that the type system with the additional higher types would be just a more complicated way of achieving the same result and would be less efficient than a system which just includes the regular types. However, there seems to be good reason to suppose that an organism that organizes its event perception in terms of such a hierarchical type system would have serious advantages over an organism that lacks the hierachical organization. These advantages include at least the following:

access and compact representation Recall from Chapter 1 that we want to consider the types that an agent has available as resources as being represented in the brain states of the agent.

Having higher types means that something corresponding to a complex type can be stored as a single element. In a complex reasoning task this can give considerable advantage in that the task can be represented in a more compact fashion and it can be easier to access (search and find) something which is a single element rather than something which is represented in terms of a complex string each element of which has to be checked in order to be sure that you have found the right element.

planning Having a compact representation facilitates planning. It is feasible to plan to take a bus trip given that we can conceive of it as such without having to plan for all the small subevents that make it up, for example, all that is involved in lifting your legs in the right way in order get on the bus. The ability to plan actions seems based on an ability to classify events in a hierarchical way.

reuse A hierarchical organization of event types means that certain event types can be reused in other event types. For example, getting on a bus (waiting for the doors to open, putting one foot inside and so on) can be very much like getting on a train. Similarly, paying for a ticket on a bus trip involves an exchange of money for a ticket in much the same way for a bus, a tram, a train, a theatre performance and so on. An agent which is not able to perceive this kind of generalization would at best use up a lot of memory coding the same event types over and over as parts of different larger event types.

learning The hierarchical organization of event types and the reuse capabilities it offers also facilitates learning of new event types. In learning to take the tram it can be useful to reuse what you have learnt about buying tickets on buses and insert it ready made into your type for tram trips. If it turns out that the procedure for buying tickets for trams is slightly different from for buses (for example, you can buy a ticket on the bus but you have to pay before you get on the tram) you nevertheless have a buying ticket type which you can modify. This might involve creating more types corresponding to those strings which the two ticket buying procedures have in common to separate out the differences between the two procedures.

Related observations about the importance of hierarchical structure for behaviour and its relationship to hierarchical reinforcement learning and neurological structure have been made for example by Botvinick (2008); Botvinick *et al.* (2009); Ribas-Fernandes *et al.* (2011).

Introducing hierarchical types in this way is an important step in our cognitive processing of events because of the computational and learning processes indicated above even if the class of events we are formally able to recognize is the same as what could be recognized by non-hierarchical regular string types, that is, technically, finite state languages. An organism with hierarchically organized types will have important advantages in acquiring new finite state event patterns. An evolutionary step from non-hierarchically organized string types to hierarchically organized types is a significant development and organisms with hierarchical types will have clear evolutionary advantages over those that do not.

However, hierarchical organization brings with it, almost as a kind of side effect, something which means that the organism could recognize classes of events that are not finite state. This is known as *recursion*. Hierarchical organization means that we can give type definitions of the form in (5).

(5) $a : T$ iff $a : T_1^\frown \ldots ^\frown T_n$

If we do not explicitly rule it out there is nothing to say that one of the T_i is not T itself. Of course, things will go badly wrong if we have a definition such as (6).

(6) $a : T$ iff $a : T_1^\frown T^\frown T_2$

If we try to perceive or create something of this type we will not be able to terminate and get into an endless string of objects of type T_1 and never be able to move on to T_2. However, if we define T in terms of a join type where at least one of the types in the join does not contain T, things will work fine. For example, (7):

(7) $a : T$ iff $a : (T_1^\frown T^\frown T_2 \vee T_1^\frown T_2)$

According to (7) anything of type T will be a string of objects of type T_1 followed by a string of equal length of objects of type T_2. It is the requirement "of equal length" which means that this type is not a regular type. For example, we could have the the regular type $T_1^{+\frown} T_2^+$ but this only expresses that we require a non-empty string of objects of type T_1 followed by a non-empty string of objects of type T_2 without the equal length requirement. What we have done here is restate a basic result from formal language theory in terms of our types. In formal language theory one talks of languages of the form $a^n b^m$ (the set of strings of n a's followed by a string of m b's, for any n and m greater than 0) which is a regular or finite state language and $a^n b^n$ (the set of strings of n a's followed by n b's, for any n greater than 0) which is context free. While this possibility of recursion is offered as soon as we allow the hierarchical typing of events in this way, it is not clear that it is exploited to a great extent in non-linguistic events. The clear examples that seem to exist are examples like opening and closing Chinese boxes, that is, boxes within boxes. The type of opening and closing (reassembling) a Chinese box could be characterized as the $a^n b^n$-type in (8).

(8) e : *OpenClose* iff
e : (*Open*$^\frown$*OpenClose*$^\frown$*Close* $\vee$ *Open*$^\frown$*Close*)

It is significant in this kind of example that the ordering of the events is forced on the agent by the physical reality of the boxes. There is only one order in which you can open all the boxes and only one order (the reverse order) in which you can close them if you are going to assemble all the boxes within a single box. It is unclear that such ordering is required in non-linguistic event types when it is not dictated by physical reality.

3.3 Syntax

We now turn our attention to how this hierarchical organization is reflected in the nature of linguistic events.[2] In Chapter 2 we used (9) as our sign type.

(9) $\left[\begin{array}{lll} \text{s-event} & : & \mathit{SEvent} \\ \text{cont} & : & \mathit{Cont} \end{array}\right]$

This represents the pairing of a speech event with content in a Saussurean sign. It does not, however, require the presence of any hierarchical information in the sign corresponding to what in linguistic theory is normally referred to as the *constituent* (or *phrase*) structure of the utterance. To some extent it is arbitrary where we add this information. We could, for example, add it under the label ‘s-event’, perhaps by dividing ‘s-event.e’ into two fields ‘phon’ and ‘syn’ (“syntax”). However, it will be more convenient (in terms of keeping paths that we need to refer to often shorter) to add a third field labelled ‘syn’ at the top level of the sign type as in (10).

(10) $\left[\begin{array}{lll} \text{s-event} & : & \mathit{SEvent} \\ \text{syn} & : & \mathit{Syn} \\ \text{cont} & : & \mathit{Cont} \end{array}\right]$

As we will see below, *Syn* will require a ‘daughters’-field for a string of signs. This means that *Sign* becomes a recursive type. It will be a *basic* type with its witnesses defined by (11).

(11) $\sigma : \mathit{Sign}$ iff $\sigma : \left[\begin{array}{lll} \text{s-event} & : & \mathit{SEvent} \\ \text{syn} & : & \mathit{Syn} \\ \text{cont} & : & \mathit{Cont} \end{array}\right]$

We shall take *Syn* to be the type (12).[3]

[2] A version of some material from this section has appeared in Cooper (2014).

[3] One might think that *Syn* should also be defined as a recursive type since it can contain *Sign* which in its turn can contain *Syn*. However, in the types we are currently proposing the only way for *Syn* to recur is through *Sign* and it is sufficient for *Sign* to be defined recursively to ensure that we do not introduce record types that are non-well founded sets of ordered pairs. That is, we want to avoid the mathematical object which is the type being a set which contains itself. In contrast the set of witnesses for a recursive type, while it will be infinite, will be well-founded.

(12) $\left[\begin{array}{lll} \text{cat} & : & \mathit{Cat} \\ \text{daughters} & : & \mathit{Sign}^* \end{array}\right]$

The type *Sign*, as so far defined, can be seen as a *universal resource*. By this we mean that it is a type which is available for all languages. *Cat* is the type of names of syntactic categories. In this chapter we will take the witnesses of *Cat* to be: 's' ("sentence"), 'np' ("noun phrase"), 'det' ("determiner"), 'n' ("noun"), 'v' ("verb") and 'vp' ("verb phrase"). These correspond to the categories we will use to cover the expressions of the fragment of English we introduced in Chapter 2. We will use capitalized versions of these category names to represent types of signs with the appropriate path in a sign type as in (13).

(13) a. *S* represents $\left[\begin{array}{l} \mathit{Sign} \\ \text{syn:}\left[\text{cat=s:}\mathit{Cat}\right] \end{array}\right]$

b. *NP* represents $\left[\begin{array}{l} \mathit{Sign} \\ \text{syn:}\left[\text{cat=np:}\mathit{Cat}\right] \end{array}\right]$

c. *Det* represents $\left[\begin{array}{l} \mathit{Sign} \\ \text{syn:}\left[\text{cat=det:}\mathit{Cat}\right] \end{array}\right]$

d. *N* represents $\left[\begin{array}{l} \mathit{Sign} \\ \text{syn:}\left[\text{cat=n:}\mathit{Cat}\right] \end{array}\right]$

e. *V* represents $\left[\begin{array}{l} \mathit{Sign} \\ \text{syn:}\left[\text{cat=v:}\mathit{Cat}\right] \end{array}\right]$

f. *VP* represents $\left[\begin{array}{l} \mathit{Sign} \\ \text{syn:}\left[\text{cat=vp:}\mathit{Cat}\right] \end{array}\right]$

Recall that the symbol $\dot{\wedge}$ represents the merge operation on types as defined in Appendix 12. This means that, for example, (13a) is the type in (14).

$$(14)\quad \left[\begin{array}{lll} \text{s-event} & : & \left[\begin{array}{lll} \text{e-loc} & : & \mathit{Loc} \\ \text{sp} & : & \mathit{Ind} \\ \text{au} & : & \mathit{Ind} \\ \text{e} & : & \mathit{Phon} \\ \text{c}_{\text{loc}} & : & \text{loc(e,e-loc)} \\ \text{c}_{\text{sp}} & : & \text{speaker(e,sp)} \\ \text{c}_{\text{au}} & : & \text{audience(e,au)} \end{array}\right] \\ \text{syn} & : & \left[\begin{array}{lll} \text{cat=s} & : & \mathit{Cat} \\ \text{daughters} & : & \mathit{Sign}^* \end{array}\right] \\ \text{cont} & : & \mathit{Cont} \end{array}\right]$$

We might think that the type *Cat* is a language specific resource and indeed if we were being more precise we might introduce separate types for different languages such as Cat_{eng}, Cat_{swe} and Cat_{tag} for the type of category names of English, Swedish and Tagalog respectively. However, there is a strong intuition that categories in different languages are more or less related. For example, we would not be surprised to find that the categories available for English and Swedish closely overlap (despite the fact that their internal syntactic structure differs) whereas the categories of English and Tagalog have less overlap. (See Gil, 2000 for discussion.) For this reason we assume that there is a universal resource *Cat* and that each language will have a subtype of *Cat* which specifies which of the categories are used in that particular language. This is related to the kind of view of linguistic universals as a kind of toolbox from which languages can choose which is put forward by Jackendoff (2002).

The ontological status of objects of type *Cat* as we have presented them is a little suspicious. Intuitively, categories should be subtypes of *Sign*, that is, like the types such as *S*, *NP* and so on in (13). We have identified signs belonging to these types as containing a particular object in *Cat* in their 'cat'-field. But one might try to characterize such signs in a different way, for example,as fulfilling certain conditions such as having certain kinds of daughters. However, this is not quite enough, for example, for lexical categories, which do not have daughters. We have to have a way of assigning categories to words and we need to create something in the sign-type that will indicate the arbitrary assignment of a category to a word. For want of a better solution we will introduce the category names which belong to the type *Cat* as a kind of "book-keeping" device that will identify a sign-type as being one whose witnesses belong to the category bearing that name.

The 'daughters'-field is required to be a string of signs, possibly the empty string, since the type *Sign** uses the Kleene-*, that is the type of strings of signs including the empty string, ε. (See Appendix 19.) Lexical items, that is words and phrases which are entered in the lexicon, will be related to signs which have the empty string of daughters. We will use *NoDaughters* to represent the type $\left[\text{syn:}\left[\text{daughters=}\varepsilon\text{:}\mathit{Sign}^*\right]\right]$.

If T_{phon} is a phonological type (that is, $T_{\text{phon}} \sqsubseteq \mathit{Phon}$) and T_{sign} is a sign type (that is, $T_{\text{sign}} \sqsubseteq$ *Sign*, then we shall use Lex(T_{phon}, T_{sign}) to represent (15)

(15) $((T_{\text{sign}} \mathbin{\dot{\wedge}} \left[\text{s-event:}\left[\text{e:}T_{\text{phon}}\right]\right]) \mathbin{\dot{\wedge}} \mathit{NoDaughters})$

This means, for example, that (16a) represents the type in (16b) which, after spelling out the abbreviations, can be seen to be the type in (16c).

(16) a. Lex("Dudamel", *NP*)

b. $(\begin{bmatrix} \mathit{NP} \\ \text{s-event:}\left[\text{e:``Dudamel''}\right] \end{bmatrix} \mathbin{\dot{\wedge}} \mathit{NoDaughters})$

c. $\left[\begin{array}{lll} \text{s-event} & : & \left[\begin{array}{lll} \text{e-loc} & : & \textit{Loc} \\ \text{sp} & : & \textit{Ind} \\ \text{au} & : & \textit{Ind} \\ \text{e} & : & \text{“Dudamel”} \\ \text{c}_{\text{loc}} & : & \text{loc(e,e-loc)} \\ \text{c}_{\text{sp}} & : & \text{speaker(e,sp)} \\ \text{c}_{\text{au}} & : & \text{audience(e,au)} \end{array}\right] \\ \text{syn} & : & \left[\begin{array}{lll} \text{cat=np} & : & \textit{Cat} \\ \text{daughters=}\varepsilon & : & \textit{Sign}^* \end{array}\right] \\ \text{cont} & : & \textit{Cont} \end{array}\right]$

We can think of ‘Lex’ as the function in (17)[4]

(17) λT_1:*Type*
λT_2:*Type* .
$((T_1 \mathbin{\dot\wedge} \left[\text{s-event:}\left[\text{e:}T_2\right]\right]) \mathbin{\dot\wedge}$ *NoDaughters*)

This function, which is used to create sign types for lexical items in a language, associating types with a syntactic category, can be seen as a universal resource. We can think of it as representing a (somewhat uninteresting, but nevertheless true) linguistic universal: “There can be speech events of given types which have no daughters (lexical items)”.

The lexical resources needed to cover our example fragment is given in (18).

(18) Lex(“Dudamel”, *NP*)
Lex(“Beethoven”, *NP*)
Lex(“a”, *Det*)
Lex(“composer”, *N*)
Lex(“conductor”, *N*)
Lex(“is”, *V*)
Lex(“ok”, *S*)
Lex(“aha”,*S*)

The choice of *S* for “ok” and “aha” might be seen as an arbitrary choice for the sake of this particular restricted fragment of English. We will not pursue details of the syntax of these particles here.

[4]We are using the notational convention for function application as used, for example, by Montague (1973) that if f is a function $f(a, b)$ is $f(b)(a)$.

The types in (18) belong to the specific resources required for English. This is not to say that these resources cannot be shared with other languages. Proper names like *Dudamel* and *Beethoven* have a special status in that they can be reused in any language, though often in modified form, at least in terms of the phonological type with which they are associated without this being perceived as quotation, code-switching or simply showing off that you know another language.

Resources like (18) can be exploited by action rules. If Lex(T_w, C) is one of the lexical resources available to an agent A and A judges an event e to be of type T_w, then A is licensed to update their gameboard with the type Lex(T_w, C). Intuitively, this means that if the agent hears an utterance of the word "composer", then they can conclude that they have heard a sign which has the category noun. This is the beginning of *parsing*, which we will regard as the same kind of update involved in event perception as discussed in the previous chapters. The action rule corresponding to lexical resources like (18), LEXRES, is given in (19), where we use 'T resource$_A$' to mean that the type T is a resource available to A. Here the relevant resource is a type returned by the function 'Lex'.) We will return below to how this relates to gameboard update.

(19) $$\frac{\text{Lex}(T, C)\ \text{resource}_A \qquad u :_A T}{:_A\ (\text{Lex}(T, C) \wedge\!\!\!\!.\ \left[\text{s-event:}\left[\text{e=}u\text{:}T\right]\right])}$$

(19) says that an agent with lexical resource Lex(T, C) who judges a speech event, u, to be of type T is licensed to judge that there is a sign of type Lex(T, C) whose 's-event.e'-field contains u.

Strings of utterances of words can be classified as utterances of phrases. That is, speech events are hierarchically organized into types of speech events in the way that we discussed at the beginning of this chapter. Agents have resources which allow them to reclassify a string of signs of certain types ("the daughters") into a single sign of another type ("the mother"). So for example a string of type *Det*⁀*N* can lead us to the conclusion that we have observed a sign of type *NP* whose daughters are of the type *Det*⁀*N*. The resource that allows us to do this is a rule which we will model as the function in (20a) which we will represent as (20b).

(20) a. $\lambda u : \textit{Det}^\frown \textit{N}$.
$$\begin{bmatrix} \textit{NP} \\ \text{syn:}\left[\text{daughters=}u\text{:}\textit{Det}^\frown\textit{N}\right] \end{bmatrix}$$
b. RuleDaughters(*NP*, *Det*⁀*N*)

'RuleDaughters' is to be the function in (21).

(21) λT_1:*Type*

$$\begin{array}{l}\lambda T_2{:}\textit{Type}\ .\\ \quad \lambda u{:}T_1\ .\ T_2 \mathbin{\dot\wedge} \left[\text{syn:}\left[\text{daughters=}u{:}T_1\right]\right]\end{array}$$

Thus 'RuleDaughters', if provided with a subtype of $\textit{Sign}^+$ and a subtype of *Sign* as arguments, will return a function which maps a string of signs of the first type to the second type with the restriction that the daughters field is filled by the string of signs. 'RuleDaughters' is one of a number of sign type construction operations which we will introduce as universal resources which have the property of returning a which combine signs. The action rule associated with 'RuleDaughters', CombineDaughters, is characterized in (22).

(22) $$\dfrac{f = \text{RuleDaughters}(T_{\text{mother}}, T_{\text{daughters}}) \qquad f\ \text{resource}_A \qquad u :_A T_{\text{daughters}}}{:_A f(u)}$$

This means, for example, that if you categorize a string of signs as being of type $\textit{Det}^\frown\textit{N}$ then you are licensed to conclude that there is a sign of type *NP* with the additional restriction that its daughters are u.

'RuleDaughters' takes care of the 'daughters'-field but it says nothing about the 's-event.e'-field, that is the phonological type associated with the new sign. This should be required to be the concatenation of all the 's-event.e'-fields in the daughters. If $u : T^+$ where T is a record type containing the path π, we will use $\text{concat}_i(u[i].\pi)$, the concatenation of all the values $u[i].\pi$ for each element in the string u in the order in which they occur in the string.

This definition is repeated in Appendix 19..

If s is a string of length n of records such that for each i, $0 \leq i < n$, $s[i].\pi$ is a defined path, $\underset{0\leq i<n}{\text{concat}}(s[i].\pi)$ denotes $s[0].\pi \ldots s[n-1].\pi$. We use $\text{concat}_i(s[i].\pi)$ to represent $\underset{0\leq i<\text{length}(s)}{\text{concat}}(s[i].\pi)$.

We can now formulate the function ConcatPhon as in (23)

(23) $$\begin{array}{l}\lambda u{:}\left[\text{s-event:}\left[\text{e:}\textit{Phon}\right]\right]^+ .\\ \quad \left[\ \text{s-event} \ :\ \left[\ \text{e=concat}_i(u[i].\text{s-event.e}) \ :\ \textit{Phon}\ \right]\ \right]\end{array}$$

ConcatPhon will map any string of speech events to the type of a single speech event whose phonology (that is the value of 's-event.e') is the concatenation of the phonologies of the individual speech events in the string.

We want to combine the function (23) with a function like that in (20). We do this by merging the domain types of the two functions and also merging the types that they return. This is shown in (24a) which in deference to standard linguistic notation for phrase structure rules could be represented as (24b).[5]

(24) a. $\lambda u : (\mathit{Det}^\frown N \mathbin{\underset{.}{\wedge}} \left[\text{s-event}\!:\!\left[\text{e}\!:\!\mathit{Phon}\right]\right]^+)\ .$
$$\begin{bmatrix} \mathit{NP} \\ \text{syn}\!:\!\left[\text{daughters}{=}u\!:\!\mathit{Det}^\frown N\right] \\ \text{s-event}\!:\!\left[\text{e}{=}\text{concat}_i(u[i].\text{s-event.e})\!:\!\mathit{Phon}\right] \end{bmatrix}$$

b. *NP* $\longrightarrow$ *Det N*

For any dependent types (that is, functions which return a type) of the form (25a) and (25b), (25c) (the merge of the two functions) represents the function (25d).

(25) a. $\lambda r\!:\!T_1\ .\ T_2(\!(r)\!)$

b. $\lambda r\!:\!T_3\ .\ T_4(\!(r)\!)$

c. $\lambda r\!:\!T_1\ .\ T_2(\!(r)\!) \mathbin{\underset{:}{\wedge}} \lambda r\!:\!T_3\ .\ T_4(\!(r)\!)$

d. $\lambda r\!:\!T_1 \mathbin{\underset{.}{\wedge}} T_3\ .\ T_2(\!(r)\!) \mathbin{\underset{.}{\wedge}} T_4(\!(r)\!)$

If $C, C_1, \ldots, C_n$ are category sign types as in (13) then (26a) represents (26b)

(26) a. $C \longrightarrow C_1 \ldots C_n$

b. RuleDaughters(C, $C_1{}^\frown \ldots {}^\frown C_n$) $\mathbin{\underset{:}{\wedge}}$ ConcatPhon

Thus the function in (24 can be represented in a third way as in (27).

(27) RuleDaughters(*NP*, *Det*$^\frown$*N*) $\mathbin{\underset{:}{\wedge}}$ ConcatPhon

The ability to factorize rules into components in this way enables us to build a theory of resources that will allow us to study them in isolation and also facilitates the development of theories of learning. It gives us a clue to how agents can build new rules by combining existing components in novel ways. It has implications for universality as well. For example, while the rule *NP* $\longrightarrow$ *Det N* is not universal (though it may be shared by a large number of languages), ConcatPhon

[5]Note that '$\longrightarrow$' used in the phrase structure rule in (24b) is not the same arrow as '$\rightarrow$' which is used in our notation for function types. We trust that the different contexts in which they occur will help to distinguish them.

is a universally available rule component, allbeit a trivial universal which says that you can have concatenations of speech events to make a larger speech event.

The rules associated with our small grammar are given by (28)

(28) $S \longrightarrow NP\ VP$
$NP \longrightarrow Det\ N$
$VP \longrightarrow V\ NP$

It may seem that we have done an awful lot of work to arrive at simple phrase structure rules. Some readers might wonder why it is worth all this trouble to ground the rules in a theory of events and action when what we come up with in the end is something that can be expressed in a standard notation which is one of the first things that a student of syntax learns. One reason has to do with our desire to explore the relationship between the perception and processing of non-linguistics events and speech events as discussed at the beginning of this chapter. Another reason has to do with placing natural constraints on syntax. By grounding syntactic structure in types of events we provide a motivation for the kind of discussion in Cooper (1982). An abstract syntax which proposes constituent structure which does not correspond to speech events is not grounded in the same way and thus presents a different kind of theory.

3.4 Semantics

We have so far specified our sign types in terms of phonology and syntax. Now we need to specify the content in the 'cont'-field. We shall start by accounting for the contents of the lexical items specified in (18). We consider first the common nouns *composer* and *conductor*. For each of these we introduce a predicate of arity $\langle Ind \rangle$. Our universal resources will include a function, 'SemCommonNoun' which will construct a common noun content from such a predicate, p. This is defined as in (29).

(29) $\text{SemCommonNoun}(p) = \lambda r{:}\big[\text{x}{:}\textit{Ind}\big]\ .\ \big[\ \text{e} \ :\ p(r.\text{x})\ \big]$

The function in (29) is of type $(\big[\text{x}{:}\textit{Ind}\big] \rightarrow \textit{RecType})$. That is, it is a function which maps any record containing a field labelled 'x' with an individual as value to a record type. We will abbreviate this type as *Ppty* (for "property") and we will call functions of this type *properties*. In our compositional semantics, properties will play a similar role as functions from individuals to truth values in Montague semantics, (that is, functions of type $\langle e, t \rangle$ in Montague's system). In place of individuals, we use records with an 'x'-field containing an individual. The motivation for this will become apparent later. Basically, we want to be able to restrict the function by adding additional fields to the domain type of the function.

In place of Montague's truth-values (that is, objects of Montague's type t) we use record types. Record types play the role of "propositions" in our system. Types, thought of as types of situations, can be considered as truth-bearing objects. They are true just in case there is something of the type and false otherwise, that is, if there is nothing of the type. The fact that we use the "proposition-like" objects as the results that our properties return is an essential ingredient in our intensional treatment of properties. In this way it follows in the tradition of property theory (Chierchia and Turner, 1988; Fox and Lappin, 2005) and Thomason's intensional approach to propositional attitudes (Thomason, 1980).

We can now combine the 'Lex'-function which builds sign types excluding content information with our new way of constructing common noun content. We define a function $\mathrm{Lex}_{\mathrm{CommonNoun}}$ which takes a phonological type and a predicate and returns a sign type. This is defined in (30).

(30) $\mathrm{Lex}_{\mathrm{CommonNoun}}(T_{\mathrm{phon}}, p)$ =
$\mathrm{Lex}(T_{\mathrm{phon}}, \mathit{N}) \wedge\!\!\!\!\cdot\;\; \left[\mathrm{cont{=}SemCommonNoun}(p){:}\mathit{Ppty}\right]$

Note that the type of the content required here is *Ppty*. In Chapter 2 we defined the content type *Cont* to be identical with *RecType*. Now we have to revise the definition of *Cont* to be (*RecType* ∨ *Ppty*). We will add further disjuncts to allow for more possibilities as we progress.

In order to cover the two common nouns *conductor* and *composer* we can include the sign types in (31) among our resources.

(31) a. $\mathrm{Lex}_{\mathrm{CommonNoun}}$("composer", composer)
b. $\mathrm{Lex}_{\mathrm{CommonNoun}}$("conductor", conductor)

Following Montague's (1973) original strategy we shall treat the contents of noun-phrases such as *Dudamel* or *a conductor* as being functions from properties to truth-bearing elements, that is, in our terms, record types. That is, noun-phrase contents will be of type (*Ppty*→*RecType*) which we will abbreviate as *Quant* (for "quantifier"). This means that we should now redefine the type of contents, *Cont*, as *RecType* ∨ *Ppty* ∨ *Quant*.

Some readers may find this a rather old-fashioned approach to the treatment of proper names which in proposals after Montague are analyzed as representing individuals rather than quantifiers. There is nothing in principle which rules out these more modern treatments in TTR but for now we will keep to the classical Montague approach. We will in any case change Montague's proposal in later chapters of this book.

Dudamel and *Beethoven* will receive proper name contents. The recipe for constructing a proper name content based on a particular individual a is given by SemPropName(a) as defined in (32).

(32) SemPropName(a) =
$\lambda P{:}\mathit{Ppty}\ .\ P(\big[\text{x}{=}a\big])$

We define $\text{Lex}_{\text{PropName}}$ which takes a phonological type (a name) and an individual (the referent of the name) and returns a sign type as in (33).

(33) $\text{Lex}_{\text{PropName}}(T_{\text{Phon}}, a)$ =
$\text{Lex}(T_{\text{Phon}}, \mathit{NP}) \mathbin{\dot\wedge} \big[\text{cont}{=}\text{SemPropName}(a){:}\mathit{Quant}\big]$

Resources to cover the proper names in our grammar could be as in (34) where $d, b : \mathit{Ind}$ (two individuals, Dudamel and Beethoven).

(34) a. $\text{Lex}_{\text{PropName}}$(“Dudamel”, d)
b. $\text{Lex}_{\text{PropName}}$(“Beethoven”, b)

Note that there is nothing to prevent us from constructing sign types with the same phonological type but different contents. Thus proper names are not required to be “logically proper” in the sense that there is one and only one individual which can be referred to by an utterance belonging to the phonological type. Names can be ambiguous. For example, there are many composers named Bach and Strauss. We have the means to construct sign types for all of them on an as needed basis. This sign-based approach, even though it holds closely to Montague’s original treatment of proper names, has the advantage that it does not require a proper name to be limited to a single individual for its content. See Cooper (2017b) for some further discussion.

Now that we have both properties and quantifiers let us check at this point that we are on the right track for combining them in something like the kind of way that we will need for compositional semantics. Suppose we want to combine a proper name content for *Dudamel* (35a) with the property of being a conductor (35b). The obvious way to do this is by applying the function in (35a) to the argument (35b) as represented in (35c). According to the definition of functional application in Appendix 4, (35c) is identical to (35d) which in turn is identical to (35e). In turn the dot notation for record path values shows (35e) to be identical to (35f).

(35) a. $\lambda P{:}\mathit{Ppty}\ .\ P(\big[\text{x}{=}d\big])$
b. $\lambda r{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \big[\ \text{e}\ :\ \text{conductor}(r.\text{x})\ \big]$
c. $\lambda P{:}\mathit{Ppty}\ .\ P(\big[\text{x}{=}d\big])$
$(\lambda r{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \big[\ \text{e}\ :\ \text{conductor}(r.\text{x})\ \big])$
d. $\lambda r{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \big[\ \text{e}\ :\ \text{conductor}(r.\text{x})\ \big](\big[\text{x}{=}d\big])$

e. $\left[\ \text{e} \ : \ \text{conductor}(\left[\text{x}{=}d\right].\text{x})\ \right]$

f. $\left[\ \text{e} \ : \ \text{conductor}(d)\ \right]$

This means that if we were dealing with a language like Russian where *Dudamel is a conductor* corresponds to a proper name followed by a common noun we would have a good way of combining the two contents by applying the content of the proper name to the content of the common noun.[6] However, things are not quite so straightforward in English. Here we use an indefinite article to form the noun phrase *a conductor*. We shall treat the content of indefinite articles as a function that maps properties to quantifiers involving the existential relation between properties. That is, it will be a function of type (*Ppty*→*Quant*), a type which should be added to our definition of *Cont* which now becomes *RecType* ∨ *Ppty* ∨ *Quant* ∨ (*Ppty*→*Quant*). As part of our universal resources we introduce a function 'SemIndefArt' which is defined as the function in (36).

(36) $\lambda Q{:}\mathit{Ppty}\ .$
$$\lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q & : & \mathit{Ppty} \\ \text{scope}{=}P & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{bmatrix}$$

We can also define a universal resource, $\text{Lex}_{\text{IndefArt}}$, which associates a phonological type (corresponding to an indefinite article in the language) with this content, as defined in (37).

(37) $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}}) =$
$$\text{Lex}(T_{\text{Phon}}, \mathit{Det}) \mathbin{\dot\wedge} \left[\text{cont}{=}\text{SemIndefArt}{:}(\mathit{Ppty}{\rightarrow}\mathit{Quant})\right]$$

The local resource for the English indefinite article would thus be (38).

(38) $\text{Lex}_{\text{IndefArt}}(\text{“a”})$

The compositional semantics of a noun-phrase consisting of a determiner followed by a noun will be the content of the determiner applied to the content of the noun. This is a case of *content forward application*. We define a function 'ContForwardApp', which is part of the universal resources, as in (39).

[6] An alternative would be to treat the content of a proper name as a record rather than a quantifier and apply the property to the record as in (35d). This would correspond to the treatment of proper names as individual denoting as discussed, for example, by Partee (1986).

(39) λT_1:*Type* λT_2:*Type* .
λu:$\bigl[$cont:$(T_2 \to T_1)\bigr]^\frown\bigl[$cont:$T_2\bigr]$.
$\bigl[$cont=u[0].cont(u[1].cont):$T_1\bigr]$

The intuition behind this function is that if you observe a string of two utterances, the first of which has a content of type $(T_2 \to T_1)$ and the second of which has a content of type T_2 then you are licensed to conclude that there is an utterance whose content is the result of applying the content of the first element in the string to the content of the second element of the string. (For the notation $s[n]$ representing the nth element of a string s see Appendix 19.) We can use 'ContForwardApp' to add constraints on content to a phrase structure rule as in the example in (40).

(40) (*NP* $\longrightarrow$ *Det N* $\overset{\wedge}{.}$ ContForwardApp(*Ppty*,*Quant*))

Recall from (24a) that *NP* $\longrightarrow$ *Det N* is the function (41a). ContForwardApp(*Ppty*,*Quant*) is the function (41b). Merging these two functions yields (41c).

(41) a. λu :(*Det*$^\frown$*N* $\overset{\wedge}{.}$ $\bigl[$s-event:$\bigl[$e:*Phon*$\bigr]\bigr]^+$) .

$$\begin{bmatrix} \textit{NP} \\ \text{syn:}\bigl[\text{daughters=}u\text{:}\textit{Det}^\frown\textit{N}\bigr] \\ \text{s-event:}\bigl[\text{e=concat}_i(u[i]\text{.s-event.e):}\textit{Phon}\bigr] \end{bmatrix}$$

b. λu:$\bigl[$cont:$(Ppty \to Quant)\bigr]^\frown\bigl[$cont:*Ppty*$\bigr]$.
$\bigl[$cont=u[0].cont(u[1].cont):*Quant*$\bigr]$

c. λu : (*Det*$^\frown$*N* $\overset{\wedge}{.}$ $\bigl[$s-event:$\bigl[$e:*Phon*$\bigr]\bigr]^+$)
$\overset{\wedge}{.}$ $\bigl[$cont:$(Ppty \to Quant)\bigr]^\frown\bigl[$cont:*Ppty*$\bigr]$.

$$\begin{bmatrix} \textit{NP} \\ \text{syn:}\bigl[\text{daughters=}u\text{:}\textit{Det}^\frown\textit{N}\bigr] \\ \text{s-event:}\bigl[\text{e=concat}_i(u[i]\text{.s-event.e):}\textit{Phon}\bigr] \\ \text{cont=}u\text{[0].cont(}u\text{[1].cont):}\textit{Quant} \end{bmatrix}$$

A convenient abbreviatory notation for this interpreted phrase structure rule is given in (42a) or more simply, since we can read the content types off the types *Det* and *N*, (42b).

(42) a. *NP* $\longrightarrow$ *Det N* | *Det*$'$(*N*$'$:*Ppty*):*Quant*

b. *NP* $\longrightarrow$ *Det N* | *Det*$'$(*N*$'$)

Here *Det*$'$ and *N*$'$ represent the contents of the determiner and noun.

We can represent the type (43a) using an informal diagrammatic tree notation which is common in linguistics as in (43b).[7]

(43) a.
$$\left[\begin{array}{l} \textit{NP} \\ \text{s-event:}\left[\text{e=syn.daughters[0].s-event.e}^\frown\text{syn.daughters[0].s-event.e:}\textit{Phon}\right] \\ \text{syn:}\left[\text{daughters:}\textit{Det}^\frown\textit{N}\right] \\ \text{cont=syn.daughters[0].cont(syn.daughters[1].cont):}\textit{Quant} \end{array}\right]$$

b.
$$\begin{array}{c} \textit{NP} \\ \alpha(\beta) \\ \begin{array}{cc} \textit{Det} & \textit{N} \\ \alpha & \beta \end{array} \end{array}$$

Here what is written under the category type (e.g. α, β) represents the value in the 'cont'-field.

The content of an utterance of *a conductor* will be (44a) applied to (44b), that is (44c).

(44) a. $\lambda Q{:}\textit{Ppty}\ .$
$$\lambda P{:}\textit{Ppty}\ .\left[\begin{array}{lll} \text{restr=}Q & : & \textit{Ppty} \\ \text{scope=}P & : & \textit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{array}\right]$$

b. $\lambda r{:}\left[\text{x:}\textit{Ind}\right]\ .\ \left[\ \text{e} \ : \ \text{conductor}(r\text{.x})\ \right]$

c.
$$\lambda P{:}\textit{Ppty}\ .\left[\begin{array}{lll} \text{restr=}\lambda r{:}\left[\text{x:}\textit{Ind}\right]\ .\ \left[\ \text{e} \ : \ \text{conductor}(r\text{.x})\ \right] & : & \textit{Ppty} \\ \text{scope=}P & : & \textit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{array}\right]$$

We will now look in more detail at the nature of the generalized quantifier in (44c). 'exist' is a predicate with arity $\langle$*Ppty*,*Ppty*$\rangle$, that is, it corresponds to a relation between two properties. The classical account of generalized quantifiers (Barwise and Cooper, 1981; Peters and Westerståhl, 2006, and much other literature) treats such quantifier relations as relations between sets. Here we will follow Cooper (2011, 2013a) in relating our treatment directly to the classical relation between sets, although, as argued in Cooper (2012a) based on earlier work by Keenan and Stavi (1986), there are ultimately good reasons for exploiting the intensionality of properties. If P is a property the relevant set is the set of individuals which have the property, which we will represent as $[\downarrow P]$. This is defined as in (45) where we use the notation $[\check{}T]$ to represent $\{a \mid a : T\}$.

(45) $[\downarrow P] = \{a \mid \exists r[r : \left[\text{x=}a\text{:}\textit{Ind}\right] \text{ and } [\check{}P(r)] \neq \emptyset]\}$

[7]A similar use of tree notation, though relating to typed feature structures rather than types, is used in HPSG (see, for example, Ginzburg and Sag, 2000, Chapter 2).

Following the terminology of Cooper (2011, 2013a) we will call $[\downarrow P]$ the property extension, or *P-extension*, of property P. Intuitively the property extension of P is the set of objects which have the property in some situation. Let us compute this for a particular example of a property, the property of being a dog given in (46).

(46) $\lambda r{:}\left[\text{x:}\textit{Ind}\right] . \left[\ \text{e} \ : \ \text{dog}(r.\text{x})\ \right]$

The property extension of (46) is given in (47).

(47) $\{a \mid \exists r[r{:}\left[\text{x=}a{:}\textit{Ind}\right] \text{ and}$
$\quad [\check{}\lambda r : \left[\text{x:}\textit{Ind}\right].\left[\ \text{e} \ : \ \text{dog}(r.\text{x})\ \right](r)] \neq \emptyset]\}$

By β-reduction (that is, the definition of function application) (47) is the same set as (48).

(48) $\{a \mid \exists r[r{:}\left[\text{x=}a{:}\textit{Ind}\right] \text{ and } [\check{}\left[\ \text{e} \ : \ \text{dog}(r.\text{x})\ \right]] \neq \emptyset]\}$

Since r is required to be of the type $\left[\text{x=}a{:}\textit{Ind}\right]$ we know that $r.\text{x}$ must be a. Therefore (48) is identical to (49).

(49) $\{a \mid \exists r[r{:}\left[\text{x=}a{:}\textit{Ind}\right] \text{ and } [\check{}\left[\ \text{e} \ : \ \text{dog}(a)\ \right]] \neq \emptyset]\}$

By the definition of record types, a record $\left[\text{e=}s\right]$ is of type $\left[\text{e:dog}(a)\right]$ just in case $s : \text{dog}(a)$. Therefore this type is non-empty just in case there is such an s. For this reason (49) is the same set as (50).

(50) $\{a \mid \exists r[r{:}\left[\text{x=}a{:}\textit{Ind}\right] \text{ and } \exists s[s : \text{dog}(a)]]\}$

Since r is no longer bound in the second conjunct of (50), (51) also defines the same set.

(51) $\{a \mid \exists r[r{:}\left[\text{x=}a{:}\textit{Ind}\right]] \text{ and } \exists s[s : \text{dog}(a)]\}$

Given the nature of records, there will be an r of the required type just in case a:*Ind*. Therefore we can characterize the same set as in (52).

(52) $\{a \mid a{:}\textit{Ind} \text{ and } \exists s[s : \text{dog}(a)]\}$

Finally, since the existence of a situation of type dog(a) requires the a is an individual given that the arity of 'dog' is $\langle$*Ind*$\rangle$ we can eliminate the first conjunct altogether so the minimal characterization of this set is (53).

(53) $\{a \mid \exists s[s : \text{dog}(a)]\}$

If P and Q are properties we want exist(P, Q) to be a type of situations which will be non-empty (that is, "true") just in case the P-extensions of P and Q have a non-empty overlap, that is there is some individual which has both property P and property Q. In symbols we can express this as (54).

(54) $[\check{}\text{exist}(P, Q)] \neq \emptyset$ iff $[\downarrow P] \cap [\downarrow Q] \neq \emptyset$

This places a requirement on objects which are assigned to the type 'exist(P, Q)' without actually tying down what kind of object they have to be. That is, it leaves it open as to which objects get assigned to the type, as long as they respect this requirement. It places a constraint on F in the models discussed on p. 19. We can, however, go a step further and make precise exactly which objects these should be. The intution is that a situation e should be of type exist(P, Q) just in case it is a witness (or "proof") of the fact that the "exist"-relation holds between P and Q (that is, that the P-extensions of P and Q have a non-empty overlap). We will say that a situation is such a witness just in case the P-extensions of the properties restricted to the situation in question stand in the required relation, that is, intuitively that the set of objects in the situation which have P overlaps with the set of objects in the situation which have Q. We will get at this notion by restricting properties to a particular situation (what we have called a resource situation in previous literature such as Barwise and Perry, 1983; Cooper, 1996). We will represent the restriction of property P to situation s as $P \upharpoonright s$. We take our previous example of the property of being a dog, repeated in (55a). Its restriction to the situation s is given in (55b).

(55) a. $\lambda r{:}\left[\text{x:}\textit{Ind}\right] . \left[\ \text{e} \ : \ \text{dog}(r.\text{x})\ \right]$
b. $\lambda r{:}\left[\text{x:}\textit{Ind}\right] . \left[\ \text{e}\underline{\varepsilon} s \ : \ \text{dog}(r.\text{x})\ \right]$

In (55b) the restricted field $\left[\text{e}\underline{\varepsilon} s\text{:dog}(r.\text{x})\right]$ requires that the object in 'e'-field is not only of type 'dog(r.x)' but also that it is either s itself or a component of s, that is, for some path π in s it is the object $s.\pi$.

The notion *component* is defined in (56), repeated in Appendix 11.1.

(56) An object, a, is a *component* of a record, r, in symbols, $a\varepsilon r$, just in case there is some path, π, in r such that $r.\pi = a$.

In terms of our informal proof theoretic notation this could be expressed as (57).

(57) a. $$\frac{\Gamma \vdash r \text{ record} \qquad \pi \in \text{paths(r)}}{\Gamma \vdash r.\pi \; \varepsilon \; r}$$

b. $$\frac{\Gamma \vdash r.\pi \; \varepsilon \; r}{\Gamma \vdash r \text{ record}} \qquad \frac{\Gamma \vdash r.\pi \; \varepsilon \; r}{\pi \in \text{paths}(r)}$$

Based on this notion we can define a notion of *present in a record* as in (58), repeated in Appendix 11.1.

(58) An object, a, is *present* in a record, r, in symbols, $a\underline{\varepsilon}r$, just in case either $a = r$ or $a\varepsilon r$.

We can express this in our informal proof theoretic notation as in (59).

(59) a. $$\frac{\Gamma \vdash r \text{ record}}{\Gamma \vdash r \; \underline{\varepsilon} \; r} \qquad \frac{\Gamma \vdash a \; \varepsilon \; r}{\Gamma \vdash a \; \underline{\varepsilon} \; r}$$

b. $$\frac{\Gamma \vdash a \; \underline{\varepsilon} \; r \qquad \begin{array}{c}[\Delta \vdash r' \text{ record}] \\ \vdots \\ \Delta \vdash \varphi\end{array} \qquad \begin{array}{c}[\Delta \vdash a' \; \varepsilon \; r'] \\ \vdots \\ \Delta \vdash \varphi\end{array}}{\Gamma \vdash \varphi}$$

We define a general notion of restricting a type by a record as in (60), repeated in Appendix 15.

(60) A type system $\mathbb{T}$ *has restricted types according to a set of labels* $\mathcal{L}$ if it is the case that

1. if T is a type, but not a record type, according to $\mathbb{T}$ and r is a record according to $\mathcal{L}$ and $\mathbb{T}$, then the restriction of T by r, $\rho(T, r)$, is a type according to $\mathbb{T}$.
2. $a :_{\mathbb{T}} \rho(T, r)$ iff $a\underline{\varepsilon}r$ and $a :_{\mathbb{T}} T$.

In terms of our informal prooftheoretic notation this can be expressed as (61).

(61) a. $$\frac{\Gamma, \mathcal{L} \vdash T \in \textbf{Type} - \text{RType} \qquad \Gamma, \mathcal{L} \vdash r \text{ record}}{\Gamma, \mathcal{L} \vdash \rho(T, r) \in \textbf{Type}}$$

b. $$\frac{\Gamma, \mathcal{L} \vdash r \text{ record} \qquad a \mathrel{\underline{\varepsilon}} r \qquad \Gamma, \mathcal{L} \vdash a : T}{\Gamma, \mathcal{L} \vdash a : \rho(T, r)}$$

$$\frac{\Gamma, \mathcal{L} \vdash a : \rho(T, r)}{\Gamma, \mathcal{L} \vdash r \text{ record}} \qquad \frac{\Gamma, \mathcal{L} \vdash a : \rho(T, r)}{a \mathrel{\underline{\varepsilon}} r} \qquad \frac{\Gamma, \mathcal{L} \vdash a : \rho(T, r)}{\Gamma, \mathcal{L} \vdash a : T}$$

We then generalize restriction to record types and other objects using $o \restriction r$ to represent the object o restricted to r. This is given by the definition in (62) repeated in Appendix 15.

(62) a. If T is a type but not a record type then

$$T \restriction r = \rho(T, r)$$

b. If X is a labelled set whose labels are not distinguished (as for example in ptypes, that is $\text{labels}(X)$ is a set of labels which can be used in record types)

$$\{\langle \ell_1, o_1 \rangle, \ldots, \langle \ell_n, o_n \rangle\}$$

then

$$X \restriction r = \{\langle \ell_1, o_1 \restriction r \rangle, \ldots, \langle \ell_n, o_n \restriction r \rangle\}$$

c. if o is

$$\langle \mathcal{T}, \Pi \rangle$$

where $\mathcal{T}$ is a dependent type and Π is a sequence of paths, then

$$o \restriction r = \langle \mathcal{T} \restriction r, \Pi \rangle$$

d. if $\mathcal{T}$ is a dependent type

$$\lambda v_1 : T_1 \,.\, \ldots \,.\, \lambda v_n : T_n \,.\, T(\!(v_1, \ldots, v_n)\!)$$

then

$$\mathcal{T} \restriction r = \lambda v_1 : T_1 \,.\, \ldots \,.\, \lambda v_n \,.\, T(\!(v_1, \ldots, v_n)\!) \restriction r$$

e. otherwise $o \restriction r = o$

We use a similar notation for restricted fields in record types as we do for manifest fields. That is, we represent (63a) as (63b) in the case where T is not a record type or a pair of a dependent type and a sequence of paths.

(63) a. $\begin{bmatrix} \ell & : & T \restriction r \end{bmatrix}$

b. $\begin{bmatrix} \ell \underline{\varepsilon} r & : & T \end{bmatrix}$

Now we can compute the property extension of (63b) in a similar fashion to the calculation for the non-restricted property. The property extension of (63b) is given in (64).

(64) $\{a \mid \exists r[r{:}\left[\text{x}{=}a{:}\textit{Ind}\right]$ and
$[\check{}\lambda r : \left[\text{x}{:}\textit{Ind}\right].\left[\ \text{e}\underline{\varepsilon}s \quad : \quad \text{dog}(r.\text{x})\ \right](r)] \neq \emptyset]\}$

By β-reduction (64) is the same set as (65).

(65) $\{a \mid \exists r[r{:}\left[\text{x}{=}a{:}\textit{Ind}\right]$ and $[\check{}\left[\ \text{e}\underline{\varepsilon}s \quad : \quad \text{dog}(r.\text{x})\ \right]] \neq \emptyset]\}$

Since r is required to be of the type $\left[\text{x}{=}a{:}\textit{Ind}\right]$ we know that r.x must be a. Therefore (65) is identical to (66).

(66) $\{a \mid \exists r[r{:}\left[\text{x}{=}a{:}\textit{Ind}\right]$ and $[\check{}\left[\ \text{e}\underline{\varepsilon}s \quad : \quad \text{dog}(a)\ \right]] \neq \emptyset]\}$

By the definition of record types, a record $\left[\text{e}{=}s'\right]$ is of type $\left[\text{e}\underline{\varepsilon}s{:}\text{dog}(a)\right]$ just in case $s'\underline{\varepsilon}s$ and s' : dog(a). Therefore this type is non-empty just in case there is some s' such that $s'\underline{\varepsilon}s$ and s' : dog(a). For this reason (66) is the same set as (67).

(67) $\{a \mid \exists r[r{:}\left[\text{x}{=}a{:}\textit{Ind}\right]$ and $\exists s'[s'\underline{\varepsilon}s$ and s' : dog$(a)]]\}$

Since r is no longer bound in the second conjunct of (67), (68) also defines the same set.

(68) $\{a \mid \exists r[r{:}\left[\text{x}{=}a{:}\textit{Ind}\right]]$ and $\exists s'[s'\underline{\varepsilon}s$ and s' : dog$(a)]\}$

Given the nature of records, there will be an r of the required type just in case a:*Ind*. Therefore we can characterize the same set as in (69).

(69) $\{a \mid a{:}\textit{Ind}$ and $\exists s'[s'\underline{\varepsilon}s$ and s' : dog$(a)\}$

Finally, since the existence of a situation of type dog(a) requires the a is an individual given that the arity of 'dog' is $\langle \textit{Ind} \rangle$ we can eliminate the first conjunct altogether so the minimal characterization of this set is (70).

(70) $\{a \mid \exists s'[s'\underline{\varepsilon}s$ and s' : dog$(a)\}$

Now we can use the notion of property restriction to characterize the witness condition for ptypes constructed with ‘exist’, as in (71).

(71) e : exist(P,Q) iff $[{\downarrow}P] \cap [{\downarrow}Q \upharpoonright e] \neq \emptyset$

This will have the consequence that any record of the type (72a) will be of type (72b).

(72) a. $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c} & : & \text{dog(x)} \\ \text{e} & : & \text{run(x)} \end{bmatrix}$

b. exist(λr:$\big[$x:*Ind*$\big]$. $\big[$e:dog(r.x)$\big]$, λr:$\big[$x:*Ind*$\big]$. $\big[$e:run(r.x)$\big]$)

In other words, (72a) is a subtype of (72b). We abbreviate the two properties in (72b) as ‘dog′’ and ‘run′’ respectively. Consider an arbitrary record, r, of type (72a) as given in (73).

(73) $r = \begin{bmatrix} \text{x} & = & a \\ \text{c} & = & s_1 \\ \text{e} & = & s_2 \\ \ldots & & \end{bmatrix}$

where s_1 : dog(a) and s_2 : run(a)

Given r we know that a must be a member of both $[{\downarrow}\,\text{dog}']$ and $[{\downarrow}\,\text{run}' \upharpoonright r]$ and that therefore r must be of type (72b). Therefore (72a)$\sqsubseteq$(72b). The argument does not go the other way, however: (72b)$\not\sqsubseteq$(72a). Consider the situation r in (74).

(74) $r = \begin{bmatrix} \text{e} & = & s_1 \end{bmatrix}$

where there is some situation $s \neq r$ such that s : dog(a) and s_1 : run(a)

In (74), r:(72b) but r is not of type (72a), since r does not “contain the information” that a is a dog.

Let us consider what we get when we apply the content we have for *a conductor*, (75a) (repeated from (44c)), to the property of composing, (75b). The result which would correspond to *a conductor composes* if we were to introduce *composes* as an intransitive verb in our resources, is given in (75c).

$$\text{(75) a. } \lambda P{:}\mathit{Ppty}\ .\ \left[\begin{array}{lll} \text{restr}{=}\lambda r{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \ :\ \text{conductor}(r.\text{x})\ \big] & : & \mathit{Ppty} \\ \text{scope}{=}P & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{array}\right]$$

$$\text{b. } \lambda r{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \ :\ \text{compose}(r.\text{x})\ \big]$$

$$\text{c. } \left[\begin{array}{lll} \text{restr}{=}\lambda r{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \ :\ \text{conductor}(r.\text{x})\ \big] & : & \mathit{Ppty} \\ \text{scope}{=}\lambda r{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \ :\ \text{compose}(r.\text{x})\ \big] & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{array}\right]$$

What would it mean for there to be something of type (75c)? In other words, what would be required to make the sentence *a conductor composes* true? There would have to be a record, r^*, which contains the three fields in the record in (76) and which meets the condition indicated.

$$\text{(76) } r^* = \left[\begin{array}{lll} \text{restr} & = & \lambda r{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \ :\ \text{conductor}(r.\text{x})\ \big] \\ \text{scope} & = & \lambda r{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \ :\ \text{compose}(r.\text{x})\ \big] \\ \text{e} & = & s \end{array}\right]$$

where $[\downarrow r^*.\text{restr}{\restriction}s]$ and $[\downarrow r^*.\text{scope}{\restriction}s]$ have a non-empty overlap.

This gives us a version of the classical treatment of indefinite articles as involving the existential quantifier, expressed in terms of a generalized quantifier which compares sets. There is, of course, a real and important question whether this is an appropriate content for the sentence *a conductor composes* which tends to get a generic reading something like "conductors, in general, compose". We will return to this issue in Chapter 7 where we will deal with the indefinite article in more detail. For now, we will ignore the sentence *a conductor composes* since we are not considering syntactic resources for it anyway.

We are concerned with finding a way to interpret the verb phrase *is a conductor*. Can we find a content for *is* which could be combined with the content for *a conductor* given in (44c) to produce an appropriate interpretation for the verb-phrase? Montague's (1973) strategy for assigning a content to *is* is reproduced in our terms in (77).

$$\begin{array}{l} \text{(77) } \lambda Q{:}\mathit{Quant}\ . \\ \quad \lambda r_1{:}\big[\text{x:}\mathit{Ind}\big]\ . \\ \quad\quad Q(\lambda r_2{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \left[\begin{array}{lll} \text{x}{=}r_2.\text{x},\ r_1.\text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{be(x)} \end{array}\right]) \end{array}$$

Here we use a manifest field based on a multiple singleton type (a singleton type formed from a singleton type, see Chapter 2, p. 59) to require the identity of r_1.x and r_2.x. In the 'e'-field of the type with the manifest field we use the predicate 'be' which we will take to be polymorphic with the set of arities as given in (78a). The witness condition associated with types constructed with 'be' is given in (78b).

(78) a. arity(be) = $\{\langle T\rangle \mid T \text{ is a type}\}$

b. e : be(a) iff $a\varepsilon e$

The intuition behind (78b) could be expressed as "To be is to be a component of a situation", that is, more technically, a "is" just in case there is a path, π, in some record, r, such that $r.\pi = a$.[8]

We will call (77) 'SemBe'. It will be included among the universal resources, together with the 'Lex_{be}' as defined in (79).

(79) If T_{Phon} is a phonological type, then $\text{Lex}_{\text{be}}(T_{\text{Phon}})$ is $\text{Lex}(T_{\text{Phon}}, V) \wedge\!\!\!\!\cdot\;$ $\left[\text{cont=SemBe:}(\textit{Quant}\rightarrow\textit{Ppty})\right]$

Among the lexical resources for English we have Lex_{be}("is").

Now let us see what we get when we combine (77) with the content of *a conductor*. This involves applying (77), repeated as (80a), to (44c), repeated as (80b). The result of this application is (80c).

(80) a. $\lambda \mathcal{Q}{:}\textit{Quant}\,.$
$\quad\lambda r_1{:}\left[\text{x:}\textit{Ind}\right].$
$\quad\quad\mathcal{Q}(\lambda r_2{:}\left[\text{x:}\textit{Ind}\right].\begin{bmatrix}\text{x=}r_2.\text{x},\ r_1.\text{x} & : & \textit{Ind}\\ \text{e} & : & \text{be(x)}\end{bmatrix})$

b. $\lambda P{:}\textit{Ppty}\,.\begin{bmatrix}\text{restr=}\lambda r{:}\left[\text{x:}\textit{Ind}\right].\left[\ \text{e} : \text{conductor}(r.\text{x})\ \right] & : & \textit{Ppty}\\ \text{scope=}P & : & \textit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{bmatrix}$

c. $\lambda r_1{:}\left[\text{x:}\textit{Ind}\right].$
$\quad\begin{bmatrix}\text{restr=}\lambda r{:}\left[\text{x:}\textit{Ind}\right].\left[\ \text{e} : \text{conductor}(r.\text{x})\ \right] & : & \textit{Ppty}\\ \text{scope=}\lambda r_2{:}\left[\text{x:}\textit{Ind}\right].\begin{bmatrix}\text{x=}r_2.\text{x},\ r_1.\text{x} & : & \textit{Ind}\\ \text{e} & : & \text{be(x)}\end{bmatrix} & : & \textit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{bmatrix}$

In order to obtain a content for *Dudamel is a conductor* we apply the content of *Dudamel*, (35a), repeated as (81a), to (80c), repeated as (81b), with result (81c).

[8]This might be compared with Quine's (1948) dictum: "To be is to be the value of a variable".

(81) a. $\lambda P{:}\mathit{Ppty}\ .\ P(\left[\text{x}{=}d\right])$

b. $\lambda r_1{:}\left[\text{x}{:}\mathit{Ind}\right]\ .$
$$\left[\begin{array}{lll} \text{restr}{=}\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \left[\ \text{e} \ :\ \text{conductor}(r.\text{x})\ \right] & : & \mathit{Ppty} \\ \text{scope}{=}\lambda r_2{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \left[\begin{array}{lll} \text{x}{=}r_2.\text{x},\ r_1.\text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{be(x)} \end{array}\right] & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{array}\right]$$

c. $$\left[\begin{array}{lll} \text{restr}{=}\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \left[\ \text{e} \ :\ \text{conductor}(r.\text{x})\ \right] & : & \mathit{Ppty} \\ \text{scope}{=}\lambda r_2{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \left[\begin{array}{lll} \text{x}{=}r_2.\text{x},\ d & : & \mathit{Ind} \\ \text{e} & : & \text{be(x)} \end{array}\right] & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{array}\right]$$

The type (81c) is distinct from the type (35f), repeated as (82), which we obtained by applying the content of *Dudamel* directly to the content of *conductor*.

(82) $\left[\ \text{e} \ :\ \text{conductor}(d)\ \right]$

There is, however, an equivalence that holds between (81c) and (82). The equivalence is not that they share the same set of witnesses. We can characterize the set of witnesses of (81c) and (83a) and the witnesses of (82) as (83b).

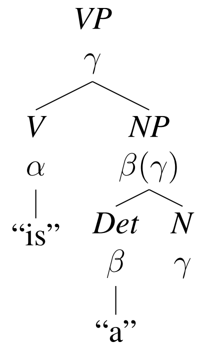
(83) a. $$\left\{ \left[\begin{array}{lll} \text{restr} & = & P \\ \text{scope} & = & Q \\ \text{e} & = & s \end{array}\right] \ \middle|\ \begin{array}{l} P = \lambda r{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \left[\ \text{e} \ :\ \text{conductor}(r.\text{x})\ \right] \\ \text{and } Q = \lambda r{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \left[\begin{array}{lll} \text{x}{=}r.\text{x},\ d & : & \mathit{Ind} \\ \text{e} & : & \text{be(x)} \end{array}\right] \\ \text{and } [{\downarrow} P{\restriction}s] \cap [{\downarrow} Q{\restriction}s] \neq \emptyset \end{array} \right\}$$

b. $\{\left[\ \text{e} \ =\ s\ \right] \mid s : \text{conductor}(d)\}$

The sets in (83) do not have any members in common. The equivalence is a weaker "truth-conditional" equivalence. (81c) has a witness ("is true") if and only if (82) has a witness. This is because the P-extensions of the property of being a conductor and the property of being identical with Dudamel can have a non-empty overlap if and only if Dudamel is a conductor. We might try to characterize the difference between the property associated with *conductor* and the property associated with *is a conductor* as "the property of being an x such that conductor(x)" and "the property of being an x such that there is a y such that conductor(y) and $y = x$". The two are truth-conditionally equivalent and for this reason in Montague's system they turn out to be the same property. For us, since we are taking a more intensional approach than Montague, they are distinct properties but they are nevertheless truth-conditionally equivalent.

Since we have two distinct properties, the question is raised whether the property that is associated with the verb-phrase should be the same as the property associated with the common noun or whether it should be the property proposed here involving existential quantification. One way to do this is to create a type corresponding to the tree in (84).

(84)



This is not compositional in the standard sense because the content of the verb phrase is not defined as some operation applied to the contents of the verb and the noun phrase, but rather it makes the content of the verb phrase be the content of the noun. Furthermore, it requires the verb and determiner utterances be of the specific types “is” and “a” respectively. This gives (84) the flavour of representing a construction type as discussed in a variety of approaches to Construction Grammar (see, for example, Boas and Sag, 2012). We can allow the type corresponding to (84) by introducing the update function (85).

(85) $$\lambda u{:}\begin{bmatrix}\textit{V}\\ \text{s-event:}\begin{bmatrix}\text{e:“is”}\end{bmatrix}\end{bmatrix} \frown \begin{bmatrix}\textit{NP}\\ \text{syn:}\begin{bmatrix}\text{daughters:}\begin{bmatrix}\textit{Det}\\ \text{s-event:}\begin{bmatrix}\text{e:“a”}\end{bmatrix}\end{bmatrix}\frown\begin{bmatrix}\textit{N}\\ \text{cont:}\textit{Ppty}\end{bmatrix}\end{bmatrix}\end{bmatrix}.\ \begin{bmatrix}\textit{VP}\\ \text{cont=}u\text{[2].syn.daughters[2].cont:}\textit{Ppty}\end{bmatrix}$$

We can call this function CnstrIsA (“is-a construction”) and merge it with $\textit{VP} \longrightarrow \textit{V NP}$. Thus one of the resources available for English is (86).

(86) $\textit{VP} \longrightarrow \textit{V NP} \underset{:}{\wedge} \text{CnstrIsA}$

This suggests that a phrase structure and construction based approach can be combined within a single framework. Since we are working with a toy fragment where the only verb is *is* and the only determiner is *a*, we can make do with (86) as the only resource for assigning content to verb-phrases. In a more general grammar we would, of course, require in addition a rule that applies the content of the verb to the content of the object noun-phrase as in (87).

(87) $\textit{VP} \longrightarrow \textit{V NP} \underset{:}{\wedge} \text{ContForwardApp}(\textit{Quant}, \textit{Ppty})$

Allowing both resources (86) and (87) simultaneously raises the issue of what the relationship should be between them. Should the more specific rule (86) take precedence and guarantee that the only content associated with the verb phrase *is a conductor* is the property which is the content of *conductor*? Or should the verb phrase be ambiguous between this interpretation and the property obtained by applying the content of *is* to the content of *a conductor*?

A more pressing issue, perhaps, is what to do about the sentence in (88).

(88) #A conductor is Dudamel

We have used the marking '#' in (88) to indicate that an utterance of this sentence would under most, if not all, circumstances be considered to be odd, though it is difficult to rule it out as ungrammatical, particularly if we are to use something corresponding to context-free phrase structure rules as we are. The oddness of (88) may have something to do with the tendency to interpret noun phrases with indefinite articles in subject position as generic as in (89).

(89) A conductor is a high-ranking individual in the musical hierarchy

(88) can be improved without becoming generic. Examples are given in (90).

(90) a. A conductor to reckon with is Dudamel
b. A conductor to consider is Dudamel
c. A conductor who impresses me as a leader in his generation is Dudamel
d. A conductor I would like to see more often in Gothenburg is Dudamel

This raises a lot of issues which we do not currently have tools to deal with. There is, however, something we can say, if we choose to allow the is-a construction interpretation of *is a conductor*. The content of the sentence *Dudamel is a conductor* on the construction analysis becomes (82), repeated as (91a), rather than (81c), repeated as (91b).

(91) a. $\left[\begin{array}{lll} \mathrm{e} & : & \mathrm{conductor}(d) \end{array}\right]$

b. $$\left[\begin{array}{lll} \mathrm{restr}{=}\lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right] . \left[\begin{array}{lll} \mathrm{e} & : & \mathrm{conductor}(r.\mathrm{x}) \end{array}\right] & : & \mathit{Ppty} \\ \mathrm{scope}{=}\lambda r_2{:}\left[\mathrm{x}{:}\mathit{Ind}\right] . \left[\begin{array}{lll} \mathrm{x}{=}r_2.\mathrm{x},\ d & : & \mathit{Ind} \\ \mathrm{e} & : & \mathrm{be}(\mathrm{x}) \end{array}\right] & : & \mathit{Ppty} \\ \mathrm{e} & : & \mathrm{exist}(\mathrm{restr},\ \mathrm{scope}) \end{array}\right]$$

If we include the resource (87) then the content of *is Dudamel* is (92a) applied to (92b), that is, (92c).

(92) a. $\lambda Q{:}\mathit{Quant}\ .$

$$\lambda r_1{:}\big[\text{x:}\mathit{Ind}\big]\ .$$

$$\mathcal{Q}(\lambda r_2{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \left[\begin{array}{lcl}\text{x=}r_2\text{.x, }r_1\text{.x} & : & \mathit{Ind}\\ \text{e} & : & \text{be(x)}\end{array}\right])$$

b. $\lambda P{:}\mathit{Ppty}\ .\ P(\big[\text{x=}d\big])$

c. $\lambda r_1{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \left[\begin{array}{lcl}\text{x=}d\text{, }r_1\text{.x} & : & \mathit{Ind}\\ \text{e} & : & \text{be(x)}\end{array}\right])$

The content of *A conductor is Dudamel* is (93a) applied to (93b) (identical with (92c)), which is (93c).

(93) a. $\lambda P{:}\mathit{Ppty}\ .\ \left[\begin{array}{lcl}\text{restr=}\lambda r{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \ :\ \text{conductor(}r\text{.x)}\ \big] & : & \mathit{Ppty}\\ \text{scope=}P & : & \mathit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{array}\right]$

b. $\lambda r_1{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{x=}d\text{, }r_1\text{.x}\ :\ \mathit{Ind}\ \big])$

c. $\left[\begin{array}{lcl}\text{restr=}\lambda r{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \ :\ \text{conductor(}r\text{.x)}\ \big] & : & \mathit{Ppty}\\ \text{scope=}\lambda r_1{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \left[\begin{array}{lcl}\text{x=}d\text{, }r_1\text{.x} & : & \mathit{Ind}\\ \text{e} & : & \text{be(x)}\end{array}\right]) & : & \mathit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{array}\right]$

(93c) is almost exactly the same type as (91b). The difference between them is that d is the first restrictor of *Ind* in (93c) whereas in (91b) it is the second restrictor. In (93c) we have, for some conductor, c, $\mathit{Ind}_{d,c}$ whereas in (91b) we have $\mathit{Ind}_{c,d}$. Thus while an analysis that only uses the content of *is* that is based on Montague's original interpretation does predict different contents for *Dudamel is a conductor* and *a conductor is Dudamel*, the difference between the types hardly seems enough to explain the difference in reaction we have to the two sentences. Given the construction analysis for *Dudamel is a conductor* we get a markedly different type (91a) which does not involve existential quantification (even though it is truth conditionally equivalent to both the types with existential quantification). The only way that (91a) can be expressed according to the resources that we have developed in this chapter is by the sentence *Dudamel is a conductor*, using the non-compositional construction 'CnstrIsA'. Thus if (91a) is the target content and we do not wish to express a content involving existential quantification, *a conductor is Dudamel* is not an option.

We thus have the beginnings of an explanation of the difference in acceptability between the two sentences. It is not the whole story since we have not explained why the quantificational readings appear odd in these cases. Note that the distinction we are making between a non-quantified reading and a reading involving an existential quantification is not available on Montague's 1973 original approach since the fact that the two contents are truth-conditionally equivalent means for Montague that they are identical. The same holds for the kind of analysis discussed in Partee (1986) where even though the content may not be built up using existential quantification the final result is still the same content that would be expressed by using existential quantification because of the truth-conditional equivalence. One might try to introduce the distinction we are making by relating utterances to an expression in an artificial logical language in addition to the content. This would correspond to the notion of logical form as discussed for example by Heim and Kratzer (1998) and much current work in linguistic semantics. The idea might be that there are two distinct logical forms such as (94) which correspond to identical contents in Montague's terms.

(94) a. conductor(dudamel)
b. $\exists x$ [conductor(x) $\wedge$ x=dudamel]

Here the challenge would be to give an explanatory account of why one expression in an artificial language, (94a), should be preferred over another, (94b), when they both express the same content. An alternative is to follow Lewis (1972) (further developed by Cresswell, 1985). The idea here is that we keep a record not only of the final content but the way in which that content is constructed – that is we keep a record of the content of each of the syntactic constituents of the English sentence and the way these contents are combined. This idea, which goes back to the notion of intensional isomorphism introduced by Carnap (1956), provides enough structure to make the distinction required here. However, there are other problems with the proposal which we will take up in Chapter 6 when we discuss intensionality.

Since acknowledgements like *aha* and *ok* do not have a specified content (*cf.* the function 'sign$_{uc}$' we used for these words in Chapter 2), we do not need a function that specifies their content but can make do with the function 'Lex' which associates them with a sign type in which the content is unspecified. Their content in a given dialogue is given by the update functions and action rules.

3.5 Incremental processing and building a chart type

If we are to take the idea that language is action seriously, we must take account of the fact that speech events happen in real time. People understand what is being said during the time that the utterance is produced. If somebody stops talking mid-sentence we are often able to complete the sentence for them or respond to what we have heard so far. There are many good accounts in the literature of why linguistic processing in *incremental* in this way. A couple of good overviews of the arguments are Ginzburg and Poesio (2016) and Kempson *et al.* (2016).

It may seem that recreating phrase structure rules and their compositional semantic interpretation in terms of a theory of action is not the best route to building a theory of the incremental nature of language processing. However, drawing this conclusion would be over-hasty. In the remainder of this chapter we will sketch a view of incrementality based on the computational technique of chart parsing which was prevalent in computational linguistics in the 1980's and 1990's. Despite the fact that it has fallen into disuse in current data-driven computational linguistics it still has significance for a theory of language as action.

For a textbook introduction to chart parsing see Jurafsky and Martin (2009), Chap. 13. The idea of a chart is that it should store all the hypotheses that we make during the processing of an utterance and allow us to compute new hypotheses to be added to the chart on the basis of what is already present in the chart. We will say that a chart is a record and we will use our resources to compute a chart type on the basis of utterance events. We will first go through an example of the incremental construction of a chart type for an agent processing an utterance of the sentence *Dudamel is a conductor*. Then we will consider what kind of update functions and action rules are needed in order to achieve this. We will, as usual, make the simplifying assumption that what we have at bottom is a string of word utterances as we are not dealing with the details of phonology. Thus we are giving a simplified view of incremental processing at the word level.

Suppose that we have so far heard an utterance of the word *Dudamel*. At this point we will say that the type of the chart is (95).

(95) $$\begin{bmatrix} \mathrm{e}_1 & : & \text{“Dudamel”} \\ \mathrm{e} & : & \big[\mathrm{e}_1\text{:start}(\Uparrow^2\mathrm{e}_1)\big]^\frown\big[\mathrm{e}_1\text{:end}(\Uparrow^2\mathrm{e}_1)\big] \end{bmatrix}$$

The main event of the chart type (represented by the 'e'-field) breaks the phonological event of type "Dudamel" down into a string of two events, the start and the end of the "Dudamel"-event.[9]

Why are the arguments to 'start' and 'end' in the string type prefixed by '$\Uparrow^2$'? Recall from the discussion on p. 45 that a string of type (96a) will be a record of type (96b).

(96) a. $\big[\mathrm{e}_1{:}T_1\big]^\frown\big[\mathrm{e}_1{:}T_2\big]$

b. $$\begin{bmatrix} \mathrm{t}_0 & : & \big[\mathrm{e}_1{:}T_1\big] \\ \mathrm{t}_1 & : & \big[\mathrm{e}_1{:}T_2\big] \end{bmatrix}$$

Thus a record of type (95) will be of the type (97).

[9]These starting and ending events correspond to what are standardly called *vertices* in the chart parsing literature.

$$(97)\quad \left[\begin{array}{lcl} \mathrm{e}_1 & : & \text{“Dudamel”} \\ \mathrm{e} & : & \left[\begin{array}{l} \mathrm{t}_0{:}\left[\mathrm{e}_1{:}\mathrm{start}(\Uparrow^2\mathrm{e}_1)\right] \\ \mathrm{t}_1{:}\left[\mathrm{e}_1{:}\mathrm{end}(\Uparrow^2\mathrm{e}_1)\right] \end{array}\right] \end{array}\right]$$

Thus the arguments to the ‘start’ and ‘end’ predicates are to be found two levels up.

Thus (95) records that we have observed an event of the phonological type “Dudamel” and an event consisting of the start of that event followed by the end of that event. Given that we have the resource the resource $\mathrm{Lex}_{\mathrm{PropName}}$(“Dudamel”, d) available, we can update (97) to (98).

$$(98)\quad \left[\begin{array}{lcl} \mathrm{e}_1 & : & \text{“Dudamel”} \\ \mathrm{e}_2 & : & \mathrm{Lex}_{\mathrm{PropName}}(\text{“Dudamel”}, d) \mathbin{\dot{\wedge}} \left[\text{s-event}{:}\left[\mathrm{e}{=}\mathrm{e}_1{:}\mathit{Phon}\right]\right] \\ \mathrm{e} & : & \left[\begin{array}{l} \mathrm{e}_1{:}\mathrm{start}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\mathrm{start}(\Uparrow^2\mathrm{e}_2) \end{array}\right] ^\frown \left[\begin{array}{l} \mathrm{e}_1{:}\mathrm{end}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\mathrm{end}(\Uparrow^2\mathrm{e}_2) \end{array}\right] \end{array}\right]$$

That is, we add the information to the chart that there is an event (labelled ‘e_2’) of the type which is the sign type corresponding to “Dudamel” and that the event which is the speech event referred to in that sign type is the utterance event, labelled by ‘e_1’. Furthermore the duration of the event labelled ‘e_2’ is the same as that labelled ‘e_1’. One could discuss where there are two events which are contemporaneous or whether there is a single utterance event which is of both types. The fact that we have presented two fields labelled ‘e_1’ and ‘e_2’ does not of itself prevent the two fields containing the same event. However, the fact that we have analyzed the sign as containing the speech event as a part (corresponding to the basic intuition that signs are pairings of utterances and contents) decides the issue for us. A sign is a record (a labelled set) which models a situation and we are not allowing sets to be members of themselves. Thus records cannot be a part of themselves.[10]

The type $\mathrm{Lex}_{\mathrm{PropName}}$(“Dudamel”, d) is a subtype of *NP*. Thus the event labelled ‘e_2’ could be the first item in a string that would be appropriate for the function which we have abbreviated as (99a) which has the type (99b).

(99) a. $S \longrightarrow NP\ VP \mid NP'(VP')$

b. $(NP^\frown VP \rightarrow \mathit{Type})$

Thus in a way that is similar to the prediction by the dog in Chapter 1 that it should run after the stick which is held up and the kind of event that this will contribute to is a game of fetch. Similarly, on observing a noun-phrase event we can predict that it might be followed by a verb

[10] ‘e_1’ and ‘e_2’ correspond to what are known as *passive edges* in the chart parsing literature. They represent information about potential constituents that have been found.

phrase event thus creating a sentence event. We will add a hypothesis event to our chart which takes place at the end of the noun-phrase event as in (100).[11]

$$(100)\quad \left[\begin{array}{lll} \mathrm{e}_1 & : & \text{“Dudamel”} \\ \mathrm{e}_2 & : & \mathrm{Lex}_{\mathrm{PropName}}(\text{“Dudamel”}, d) \mathbin{\dot{\wedge}} \left[\text{s-event:}\left[\mathrm{e}{=}\Uparrow^2\mathrm{e}_1{:}\mathit{Phon}\right]\right] \\ \mathrm{e}_3 & : & \left[\begin{array}{l} \text{rule}{=}S \longrightarrow \mathit{NP}\ \mathit{VP} \mid \mathit{NP}'(\mathit{VP}'){:}(\mathit{NP}^\frown \mathit{VP} \to \mathit{Type}) \\ \text{fnd}{=}\Uparrow\mathrm{e}_2{:}\mathit{Sign} \\ \text{req}{=}\mathit{VP}{:}\mathit{Type} \\ \text{e:required(req,rule)} \end{array}\right] \\ \mathrm{e} & : & \left[\begin{array}{l} \mathrm{e}_1{:}\text{start}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\text{start}(\Uparrow^2\mathrm{e}_2) \end{array}\right]^\frown \left[\begin{array}{l} \mathrm{e}_1{:}\text{end}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\text{end}(\Uparrow^2\mathrm{e}_2) \\ \mathrm{e}_3{:}\text{start}(\Uparrow^3\mathrm{e}_3)^\frown\text{end}(\Uparrow^3\mathrm{e}_3) \end{array}\right] \end{array}\right]$$

In the e_3-field the ‘rule’-field is for a syntactic rule, that is, a function from a string of signs of a given type to a type. The ‘fnd’-field is for a sign or string of signs so far found which match an initial segment of a string of the type required by the rule. The ‘req’-field is the type of the remaining string required to satisfy the rule as expressed in the ‘e’-field. This hypothesis event both starts and ends at the end of the event of the noun-phrase event e_2.[12]

We can now progress to the next word in the input string as shown in (101).

$$(101)\quad \left[\begin{array}{lll} \mathrm{e}_1 & : & \text{“Dudamel”} \\ \mathrm{e}_2 & : & \mathrm{Lex}_{\mathrm{PropName}}(\text{“Dudamel”}, d) \mathbin{\dot{\wedge}} \left[\text{s-event:}\left[\mathrm{e}{=}\Uparrow^2\mathrm{e}_1{:}\mathit{Phon}\right]\right] \\ \mathrm{e}_3 & : & \left[\begin{array}{l} \text{rule}{=}S \longrightarrow \mathit{NP}\ \mathit{VP} \mid \mathit{NP}'(\mathit{VP}'){:}(\mathit{NP}^\frown \mathit{VP} \to \mathit{Type}) \\ \text{fnd}{=}\Uparrow\mathrm{e}_2{:}\mathit{Sign} \\ \text{req}{=}\mathit{VP}{:}\mathit{Type} \\ \text{e:required(req,rule)} \end{array}\right] \\ \mathrm{e}_4 & : & \text{“is”} \\ \mathrm{e} & : & \left[\begin{array}{l} \mathrm{e}_1{:}\text{start}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\text{start}(\Uparrow^2\mathrm{e}_2) \end{array}\right]^\frown \left[\begin{array}{l} \mathrm{e}_1{:}\text{end}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\text{end}(\Uparrow^2\mathrm{e}_2) \\ \mathrm{e}_3{:}\text{start}(\Uparrow^3\mathrm{e}_3)^\frown\text{end}(\Uparrow^3\mathrm{e}_3) \\ \mathrm{e}_4{:}\text{start}(\Uparrow^2\mathrm{e}_4) \end{array}\right]^\frown \left[\mathrm{e}_4{:}\text{end}(\Uparrow^2\mathrm{e}_4)\right] \end{array}\right]$$

Note that the start of the “is”-event is aligned with the end of “Dudamel”-event. This allows for the fact that there is no break between the words and that the exact pronunciation of the final /l/

[11] In terms of the traditional chart parsing terminology this corresponds to an *active edge* involving a *dotted rule*. The fact that the addition of this type to the chart type is triggered by finding something of an appropriate type to be the leftmost element in a string the would be an appropriate argument to the rule corresponds to what is called a *left-corner* parsing strategy.

[12] With respect to the word string event labelled by ‘e’, it is a *punctual* event.

(102)

$$
\left[\begin{array}{lll}
\mathrm{e}_1 & : & \text{“Dudamel”} \\
\mathrm{e}_2 & : & \mathrm{Lex}_{\mathrm{PropName}}(\text{“Dudamel”}, d) \mathbin{\dot{\wedge}} \left[\text{s-event:}\left[\mathrm{e}{=}\Uparrow^2\mathrm{e}_1{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_3 & : & \left[\begin{array}{l}
\text{rule}{=}S \longrightarrow NP\ VP \mid NP'(VP'){:}(NP^\frown VP \rightarrow \mathit{Type}) \\
\text{fnd}{=}\Uparrow\mathrm{e}_2{:}\mathit{Sign} \\
\text{req}{=}VP{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\mathrm{e}_4 & : & \text{“is”} \\
\mathrm{e}_5 & : & \mathrm{Lex}_{\mathrm{be}}(\text{“is”}) \mathbin{\dot{\wedge}} \left[\text{s-event:}\left[\mathrm{e}{=}\Uparrow^2\mathrm{e}_4{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_6 & : & \left[\begin{array}{l}
\text{rule}{=}VP \longrightarrow [_V\ \text{“is”}]\ [_{NP}\ [_{Det}\ \text{“a”}]\ N] \mid N'{:} \\
\qquad (V \mathbin{\dot{\wedge}} \left[\text{s-event:}\left[\text{e:“is”}\right]\right]^\frown \\
\qquad NP \mathbin{\dot{\wedge}} \left[\text{syn:}\left[\begin{array}{l}\text{daughters:}\mathit{Det} \mathbin{\dot{\wedge}} \left[\text{s-event:}\left[\text{e:“a”}\right]\right] \\ \qquad {}^\frown N \mathbin{\dot{\wedge}} \left[\text{cont:}\mathit{Ppty}\right]\end{array}\right]\right] \\
\qquad \rightarrow \mathit{Type}) \\
\text{fnd}{=}\Uparrow\mathrm{e}_5{:}\mathit{Sign} \\
\text{req}{=}NP{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\mathrm{e} & : & \left[\begin{array}{l}\mathrm{e}_1{:}\text{start}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\text{start}(\Uparrow^2\mathrm{e}_2)\end{array}\right]^\frown
\left[\begin{array}{l}\mathrm{e}_1{:}\text{end}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\text{end}(\Uparrow^2\mathrm{e}_2) \\ \mathrm{e}_3{:}\text{start}(\Uparrow^3\mathrm{e}_3)^\frown\text{end}(\Uparrow^3\mathrm{e}_3) \\ \mathrm{e}_4{:}\text{start}(\Uparrow^2\mathrm{e}_4) \\ \mathrm{e}_5{:}\text{start}(\Uparrow^2\mathrm{e}_5)\end{array}\right]^\frown
\left[\begin{array}{l}\mathrm{e}_4{:}\text{end}(\Uparrow^2\mathrm{e}_4) \\ \mathrm{e}_5{:}\text{end}(\Uparrow^2\mathrm{e}_5) \\ \mathrm{e}_6{:}\text{start}(\Uparrow^3\mathrm{e}_6)^\frown\text{end}(\Uparrow^3\mathrm{e}_6)\end{array}\right]
\end{array}\right]
$$

in “Dudamel” is influenced by the pronuniciation of the initial /i/ in “is” through coarticulation.[13]

We can now go through similar procedures as we did for *Dudamel* adding both a lexical event based on our lexical resources and a hypothesis event based on the only rule for strings beginning with a V that we have in our resources. The result of these two steps is given in (102).

Now we can add *a* and *conductor* in a similar way with the result shown in (103).

Note that there is no possibility of adding a hypothesis event based on the utterance of *conductor* given the resources we have since our small grammar does not include a phrase structure rule for strings whose first element is of type *N*. However, now for the first time we have found something which fulfills one of our hypotheses. The hypothesis event labelled ‘e_9’ has the type *N* in its ‘req’-field. The event labelled ‘e_{11}’ is required to be of a subtype of *N* and thus fulfils the requirement of ‘e_9’. Furthermore, the start of e_{11} is aligned with the end (and also the start)

[13] It also means that the number of elements in the string labelled ‘e’ is the same as the number of vertices in a standard chart.

(103)

$$
\left[\begin{array}{lll}
\text{e}_1 & : & \text{“Dudamel”} \\
\text{e}_2 & : & \text{Lex}_{\text{PropName}}(\text{“Dudamel”}, d) \underset{.}{\wedge} \left[\text{s-event:}\left[\text{e}{=}{\Uparrow^2}\text{e}_1{:}\mathit{Phon}\right]\right] \\
\text{e}_3 & : & \left[\begin{array}{l}
\text{rule}{=}\mathit{S} \longrightarrow \mathit{NP}\ \mathit{VP} \mid \mathit{NP}'(\mathit{VP}'){:}(\mathit{NP}{\frown}\mathit{VP} \to \mathit{Type}) \\
\text{fnd}{=}{\Uparrow}\text{e}_2{:}\mathit{Sign} \\
\text{req}{=}\mathit{VP}{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\text{e}_4 & : & \text{“is”} \\
\text{e}_5 & : & \text{Lex}_{\text{be}}(\text{“is”}) \underset{.}{\wedge} \left[\text{s-event:}\left[\text{e}{=}{\Uparrow^2}\text{e}_4{:}\mathit{Phon}\right]\right] \\
\text{e}_6 & : & \left[\begin{array}{l}
\text{rule}{=}\mathit{VP} \longrightarrow [_{V}\ \text{“is”}]\ [_{NP}\ [_{Det}\ \text{“a”}]\ \mathit{N}] \mid \mathit{N}'{:} \\
\qquad (\mathit{V} \underset{.}{\wedge} \left[\text{s-event:}\left[\text{e:“is”}\right]\right] {\frown} \\
\qquad \mathit{NP} \underset{.}{\wedge} \left[\text{syn:}\left[\begin{array}{l}\text{daughters:}\mathit{Det} \underset{.}{\wedge} \left[\text{s-event:}\left[\text{e:“a”}\right]\right] \\ \qquad {\frown}\ \mathit{N} \underset{.}{\wedge} \left[\text{cont:}\mathit{Ppty}\right]\end{array}\right]\right] \\
\qquad \to \mathit{Type}) \\
\text{fnd}{=}{\Uparrow}\text{e}_5{:}\mathit{Sign} \\
\text{req}{=}\mathit{NP}{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\text{e}_7 & : & \text{“a”} \\
\text{e}_8 & : & \text{Lex}_{\text{IndefArt}}(\text{“a”}) \underset{.}{\wedge} \left[\text{s-event:}\left[\text{e}{=}{\Uparrow^2}\text{e}_7{:}\mathit{Phon}\right]\right] \\
\text{e}_9 & : & \left[\begin{array}{l}
\text{rule}{=}\mathit{NP} \longrightarrow \mathit{Det}\ \mathit{N} \mid \mathit{Det}'(\mathit{N}'){:}(\mathit{Det}{\frown}\mathit{N} \to \mathit{Type}) \\
\text{fnd}{=}{\Uparrow}\text{e}_8{:}\mathit{Sign} \\
\text{req}{=}\mathit{N}{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\text{e}_{10} & : & \text{“conductor”} \\
\text{e}_{11} & : & \text{Lex}_{\text{CommonNoun}}(\text{“conductor”}, \text{conductor}) \underset{.}{\wedge} \left[\text{s-event:}\left[\text{e}{=}{\Uparrow^2}\text{e}_{10}{:}\mathit{Phon}\right]\right] \\
\text{e} & : & \left[\begin{array}{l}
\text{e}_1{:}\text{start}({\Uparrow^2}\text{e}_1) \\
\text{e}_2{:}\text{start}({\Uparrow^2}\text{e}_2)
\end{array}\right] {\frown} \left[\begin{array}{l}
\text{e}_1{:}\text{end}({\Uparrow^2}\text{e}_1) \\
\text{e}_2{:}\text{end}({\Uparrow^2}\text{e}_2) \\
\text{e}_3{:}\text{start}({\Uparrow^3}\text{e}_3){\frown}\text{end}({\Uparrow^3}\text{e}_3) \\
\text{e}_4{:}\text{start}({\Uparrow^2}\text{e}_4) \\
\text{e}_5{:}\text{start}({\Uparrow^2}\text{e}_5)
\end{array}\right] {\frown} \left[\begin{array}{l}
\text{e}_4{:}\text{end}({\Uparrow^2}\text{e}_4) \\
\text{e}_5{:}\text{end}({\Uparrow^2}\text{e}_5) \\
\text{e}_6{:}\text{start}({\Uparrow^3}\text{e}_6){\frown}\text{end}({\Uparrow^3}\text{e}_6) \\
\text{e}_7{:}\text{start}({\Uparrow^2}\text{e}_7) \\
\text{e}_8{:}\text{start}({\Uparrow^2}\text{e}_8)
\end{array}\right] {\frown} \\
& & \qquad \left[\begin{array}{l}
\text{e}_7{:}\text{end}({\Uparrow^2}\text{e}_7) \\
\text{e}_8{:}\text{end}({\Uparrow^2}\text{e}_8) \\
\text{e}_9{:}\text{start}({\Uparrow^3}\text{e}_9){\frown}\text{end}({\Uparrow^3}\text{e}_9) \\
\text{e}_{10}{:}\text{start}({\Uparrow^2}\text{e}_{10}) \\
\text{e}_{11}{:}\text{start}({\Uparrow^2}\text{e}_{11})
\end{array}\right] {\frown} \left[\begin{array}{l}
\text{e}_{10}{:}\text{end}({\Uparrow^2}\text{e}_{10}) \\
\text{e}_{11}{:}\text{end}({\Uparrow^2}\text{e}_{11})
\end{array}\right]
\end{array}\right]
$$

of ‘e_9’. This means that we can update the chart-type by adding a new field for an event of the type returned by applying ‘e_9.rule’ (a function) to the string $e_9.\text{fnd}^\frown e_{11}$. The start of this new *NP*-event will be aligned with the start of e_9.fnd (that is, e_8). The end of the new event is aligned with the end of e_{11}. The resulting chart-type is given in (104).

The event labelled ‘e_{12}’ will be of type *NP* and thus satisfy the requirement of e_6. By carrying out the same procedure as before we will obtain a new event (labelled ‘e_{13}’) of type *VP* which will satisfy the requirement of ‘e_3’ which will allow us to add a new event (labelled ‘e_{14}’) of type *S* whose start is at the beginning of the string labelled ‘e’ and whose end is at the end of that string. The final chart type is given in (105).

(104)

$$
\left[\begin{array}{lll}
\mathrm{e}_1 & : & \text{“Dudamel”} \\
\mathrm{e}_2 & : & \mathrm{Lex}_{\mathrm{PropName}}(\text{“Dudamel”}, d) \mathbin{\dot\wedge} \left[\text{s-event}{:}\left[\mathrm{e}{=}\Uparrow^2\mathrm{e}_1{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_3 & : & \left[\begin{array}{l}
\text{rule}{=}\mathit{S} \longrightarrow \mathit{NP}\ \mathit{VP} \mid \mathit{NP}'(\mathit{VP}'){:}(\mathit{NP}^\frown \mathit{VP} \rightarrow \mathit{Type}) \\
\text{fnd}{=}\Uparrow\mathrm{e}_2{:}\mathit{Sign} \\
\text{req}{=}\mathit{VP}{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\mathrm{e}_4 & : & \text{“is”} \\
\mathrm{e}_5 & : & \mathrm{Lex}_{\mathrm{be}}(\text{“is”}) \mathbin{\dot\wedge} \left[\text{s-event}{:}\left[\mathrm{e}{=}\Uparrow^2\mathrm{e}_4{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_6 & : & \left[\begin{array}{l}
\text{rule}{=}\mathit{VP} \longrightarrow [_{V}\ \text{“is”}]\ [_{NP}\ [_{Det}\ \text{“a”}]\ \mathit{N}] \mid \mathit{N}'{:} \\
\qquad\qquad (\mathit{V} \mathbin{\dot\wedge} \left[\text{s-event}{:}\left[\text{e:“is”}\right]\right]^\frown \\
\qquad\qquad \mathit{NP} \mathbin{\dot\wedge} \left[\text{syn}{:}\left[\begin{array}{l}\text{daughters}{:}\mathit{Det} \mathbin{\dot\wedge} \left[\text{s-event}{:}\left[\text{e:“a”}\right]\right] \\ \qquad {}^\frown\ \mathit{N} \mathbin{\dot\wedge} \left[\text{cont}{:}\mathit{Ppty}\right]\end{array}\right]\right] \\
\qquad\qquad \rightarrow \mathit{Type}) \\
\text{fnd}{=}\Uparrow\mathrm{e}_5{:}\mathit{Sign} \\
\text{req}{=}\mathit{NP}{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\mathrm{e}_7 & : & \text{“a”} \\
\mathrm{e}_8 & : & \mathrm{Lex}_{\mathrm{IndefArt}}(\text{“a”}) \mathbin{\dot\wedge} \left[\text{s-event}{:}\left[\mathrm{e}{=}\Uparrow^2\mathrm{e}_7{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_9 & : & \left[\begin{array}{l}
\text{rule}{=}\mathit{NP} \longrightarrow \mathit{Det}\ \mathit{N} \mid \mathit{Det}'(\mathit{N}'){:}(\mathit{Det}^\frown \mathit{N} \rightarrow \mathit{Type}) \\
\text{fnd}{=}\Uparrow\mathrm{e}_8{:}\mathit{Sign} \\
\text{req}{=}\mathit{N}{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\mathrm{e}_{10} & : & \text{“conductor”} \\
\mathrm{e}_{11} & : & \mathrm{Lex}_{\mathrm{CommonNoun}}(\text{“conductor”}, \text{conductor}) \mathbin{\dot\wedge} \left[\text{s-event}{:}\left[\mathrm{e}{=}\Uparrow^2\mathrm{e}_{10}{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_{12} & : & \mathrm{e}_9.\text{rule}(\mathrm{e}_9.\text{fnd}^\frown \mathrm{e}_{11}) \\
\mathrm{e} & : & \left[\begin{array}{l}\mathrm{e}_1{:}\text{start}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\text{start}(\Uparrow^2\mathrm{e}_2)\end{array}\right]^\frown
\left[\begin{array}{l}\mathrm{e}_1{:}\text{end}(\Uparrow^2\mathrm{e}_1) \\ \mathrm{e}_2{:}\text{end}(\Uparrow^2\mathrm{e}_2) \\ \mathrm{e}_3{:}\text{start}(\Uparrow^3\mathrm{e}_3)^\frown\text{end}(\Uparrow^3\mathrm{e}_3) \\ \mathrm{e}_4{:}\text{start}(\Uparrow^2\mathrm{e}_4) \\ \mathrm{e}_5{:}\text{start}(\Uparrow^2\mathrm{e}_5)\end{array}\right]^\frown
\left[\begin{array}{l}\mathrm{e}_4{:}\text{end}(\Uparrow^2\mathrm{e}_4) \\ \mathrm{e}_5{:}\text{end}(\Uparrow^2\mathrm{e}_5) \\ \mathrm{e}_6{:}\text{start}(\Uparrow^3\mathrm{e}_6)^\frown\text{end}(\Uparrow^3\mathrm{e}_6) \\ \mathrm{e}_7{:}\text{start}(\Uparrow^2\mathrm{e}_7) \\ \mathrm{e}_8{:}\text{start}(\Uparrow^2\mathrm{e}_8) \\ \mathrm{e}_{12}{:}\text{start}(\Uparrow^2\mathrm{e}_{12})\end{array}\right]^\frown \\
 & & \left[\begin{array}{l}\mathrm{e}_7{:}\text{end}(\Uparrow^2\mathrm{e}_7) \\ \mathrm{e}_8{:}\text{end}(\Uparrow^2\mathrm{e}_8) \\ \mathrm{e}_9{:}\text{start}(\Uparrow^3\mathrm{e}_9)^\frown\text{end}(\Uparrow^3\mathrm{e}_9) \\ \mathrm{e}_{10}{:}\text{start}(\Uparrow^2\mathrm{e}_{10}) \\ \mathrm{e}_{11}{:}\text{start}(\Uparrow^2\mathrm{e}_{11})\end{array}\right]^\frown
\left[\begin{array}{l}\mathrm{e}_{10}{:}\text{end}(\Uparrow^2\mathrm{e}_{10}) \\ \mathrm{e}_{11}{:}\text{end}(\Uparrow^2\mathrm{e}_{11}) \\ \mathrm{e}_{12}{:}\text{end}(\Uparrow^2\mathrm{e}_{12})\end{array}\right]
\end{array}\right]
$$

(105)

$$
\left[\begin{array}{lcl}
\mathrm{e}_1 & : & \text{“Dudamel”} \\
\mathrm{e}_2 & : & \mathrm{Lex}_{\mathrm{PropName}}(\text{“Dudamel”}, d) \underset{\cdot}{\wedge} \left[\text{s-event:} \left[\mathrm{e}{=}{\Uparrow^2}\mathrm{e}_1{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_3 & : & \left[\begin{array}{l}
\text{rule}{=}\mathit{S} \longrightarrow \mathit{NP}\ \mathit{VP} \mid \mathit{NP}'(\mathit{VP}'){:}(\mathit{NP}{\frown}\mathit{VP} \to \mathit{Type}) \\
\text{fnd}{=}{\Uparrow}\mathrm{e}_2{:}\mathit{Sign} \\
\text{req}{=}\mathit{VP}{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\mathrm{e}_4 & : & \text{“is”} \\
\mathrm{e}_5 & : & \mathrm{Lex}_{\mathrm{be}}(\text{“is”}) \underset{\cdot}{\wedge} \left[\text{s-event:} \left[\mathrm{e}{=}{\Uparrow^2}\mathrm{e}_4{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_6 & : & \left[\begin{array}{l}
\text{rule}{=}\mathit{VP} \longrightarrow [_{V}\ \text{“is”}]\ [_{\mathit{NP}}\ [_{\mathit{Det}}\ \text{“a”}]\ \mathit{N}] \mid \mathit{N}'{:} \\
\qquad (\mathit{V} \underset{\cdot}{\wedge} \left[\text{s-event:} \left[\text{e:“is”}\right]\right] {\frown} \\
\qquad \mathit{NP} \underset{\cdot}{\wedge} \left[\text{syn:} \left[\begin{array}{l}\text{daughters:}\mathit{Det} \underset{\cdot}{\wedge} \left[\text{s-event:} \left[\text{e:“a”}\right]\right] \\ \qquad {\frown}\ \mathit{N} \underset{\cdot}{\wedge} \left[\text{cont:}\mathit{Ppty}\right]\end{array}\right]\right] \\
\qquad \to \mathit{Type}) \\
\text{fnd}{=}{\Uparrow}\mathrm{e}_5{:}\mathit{Sign} \\
\text{req}{=}\mathit{NP}{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\mathrm{e}_7 & : & \text{“a”} \\
\mathrm{e}_8 & : & \mathrm{Lex}_{\mathrm{IndefArt}}(\text{“a”}) \underset{\cdot}{\wedge} \left[\text{s-event:} \left[\mathrm{e}{=}{\Uparrow^2}\mathrm{e}_7{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_9 & : & \left[\begin{array}{l}
\text{rule}{=}\mathit{NP} \longrightarrow \mathit{Det}\ \mathit{N} \mid \mathit{Det}'(\mathit{N}'){:}(\mathit{Det}{\frown}\mathit{N} \to \mathit{Type}) \\
\text{fnd}{=}{\Uparrow}\mathrm{e}_8{:}\mathit{Sign} \\
\text{req}{=}\mathit{N}{:}\mathit{Type} \\
\text{e:required(req,rule)}
\end{array}\right] \\
\mathrm{e}_{10} & : & \text{“conductor”} \\
\mathrm{e}_{11} & : & \mathrm{Lex}_{\mathrm{CommonNoun}}(\text{“conductor”}, \text{conductor}) \underset{\cdot}{\wedge} \left[\text{s-event:} \left[\mathrm{e}{=}{\Uparrow^2}\mathrm{e}_{10}{:}\mathit{Phon}\right]\right] \\
\mathrm{e}_{12} & : & \mathrm{e}_9.\text{rule}(\mathrm{e}_9.\text{fnd}{\frown}\mathrm{e}_{11}) \\
\mathrm{e}_{13} & : & \mathrm{e}_6.\text{rule}(\mathrm{e}_6.\text{fnd}{\frown}\mathrm{e}_{12}) \\
\mathrm{e}_{14} & : & \mathrm{e}_3.\text{rule}(\mathrm{e}_3.\text{fnd}{\frown}\mathrm{e}_{13}) \\
\mathrm{e} & : & \left[\begin{array}{l}
\mathrm{e}_1{:}\text{start}({\Uparrow^2}\mathrm{e}_1) \\
\mathrm{e}_2{:}\text{start}({\Uparrow^2}\mathrm{e}_2) \\
\mathrm{e}_{14}{:}\text{start}({\Uparrow^2}\mathrm{e}_{14})
\end{array}\right] {\frown}
\left[\begin{array}{l}
\mathrm{e}_1{:}\text{end}({\Uparrow^2}\mathrm{e}_1) \\
\mathrm{e}_2{:}\text{end}({\Uparrow^2}\mathrm{e}_2) \\
\mathrm{e}_3{:}\text{start}({\Uparrow^3}\mathrm{e}_3){\frown}\text{end}({\Uparrow^3}\mathrm{e}_3) \\
\mathrm{e}_4{:}\text{start}({\Uparrow^2}\mathrm{e}_4) \\
\mathrm{e}_5{:}\text{start}({\Uparrow^2}\mathrm{e}_5) \\
\mathrm{e}_{13}{:}\text{start}({\Uparrow^2}\mathrm{e}_{13})
\end{array}\right] {\frown}
\left[\begin{array}{l}
\mathrm{e}_4{:}\text{end}({\Uparrow^2}\mathrm{e}_4) \\
\mathrm{e}_5{:}\text{end}({\Uparrow^2}\mathrm{e}_5) \\
\mathrm{e}_6{:}\text{start}({\Uparrow^3}\mathrm{e}_6){\frown}\text{end}({\Uparrow^3}\mathrm{e}_6) \\
\mathrm{e}_7{:}\text{start}({\Uparrow^2}\mathrm{e}_7) \\
\mathrm{e}_8{:}\text{start}({\Uparrow^2}\mathrm{e}_8) \\
\mathrm{e}_{12}{:}\text{start}({\Uparrow^2}\mathrm{e}_{12})
\end{array}\right] {\frown} \\
& & \qquad \left[\begin{array}{l}
\mathrm{e}_7{:}\text{end}({\Uparrow^2}\mathrm{e}_7) \\
\mathrm{e}_8{:}\text{end}({\Uparrow^2}\mathrm{e}_8) \\
\mathrm{e}_9{:}\text{start}({\Uparrow^3}\mathrm{e}_9){\frown}\text{end}({\Uparrow^3}\mathrm{e}_9) \\
\mathrm{e}_{10}{:}\text{start}({\Uparrow^2}\mathrm{e}_{10}) \\
\mathrm{e}_{11}{:}\text{start}({\Uparrow^2}\mathrm{e}_{11})
\end{array}\right] {\frown}
\left[\begin{array}{l}
\mathrm{e}_{10}{:}\text{end}({\Uparrow^2}\mathrm{e}_{10}) \\
\mathrm{e}_{11}{:}\text{end}({\Uparrow^2}\mathrm{e}_{11}) \\
\mathrm{e}_{12}{:}\text{end}({\Uparrow^2}\mathrm{e}_{12}) \\
\mathrm{e}_{13}{:}\text{end}({\Uparrow^2}\mathrm{e}_{13}) \\
\mathrm{e}_{14}{:}\text{end}({\Uparrow^2}\mathrm{e}_{14})
\end{array}\right]
\end{array}\right]
$$

We now need to turn our attention to the action rules that will achieve this building of the chart type. We will introduce a field 'current-utterance' into the field 'shared' on the gameboard. This field will be used for the incremental construction of a chart during the course of an utterance. The new type *InfoState* (which is a modification of the *InfoState* as defined in Chapter 2, example (88), p. 83) is given in (106).

(106) $$\left[\begin{array}{lll} \text{private} & : & \left[\begin{array}{lll} \text{agenda} & : & \text{list}(\textit{RecType}) \end{array}\right] \\ \text{shared} & : & \left[\begin{array}{lll} \text{latest-utterance} & : & \textit{Sign}^{*} \\ \text{current-utterance} & : & \textit{RecType} \\ \text{commitments} & : & \textit{RecType} \end{array}\right] \end{array}\right]$$

The initial type *InitInfoState* is now (107).

(107) $$\left[\begin{array}{lll} \text{private} & : & \left[\begin{array}{lll} \text{agenda=[ ]} & : & \text{list}(\textit{RecType}) \end{array}\right] \\ \text{shared} & : & \left[\begin{array}{lll} \text{latest-utterance=}\varepsilon & : & \textit{Sign}^{*} \\ \text{current-utterance=}\textit{ERec} & : & \textit{RecType} \\ \text{commitments=}\textit{Rec} & : & \textit{RecType} \end{array}\right] \end{array}\right]$$

We first characterize an update function, $\mathrm{f}_{\text{INTEGLEX}}$, and action rule, INTEGLEX, for integrating lexical events into the chart. In order to do this we will need to introduce two new technical notions. First we will need to find the maximum i such 'e_i' is among the labels in the current chart type, so that we can make sure that what we add to the chart type will have the label 'e_{i+1}'. If X is a labelled set, we say that $\max_\ell(X)$ is the number defined in (108).

(108) $\max(\{i \mid \ell_i \in \text{labels}(X)\})$

For convenience we abbreviate (109a) as (109b), where 'incr' stands for "increment".

(109) a. $\max_\ell(X) + 1$

b. $\text{incr}_\ell(X)$

Secondly, we introduce an operation of *concatenative merge*, '$\wedge_{\text{concat}}$', which is exactly like ordinary merge except in the case of two string types where $T_1 \wedge_{\text{concat}} T_2$ is $T_1{}^\frown T_2$.

We can now define '$\mathrm{f}_{\text{INTEGLEX}}$' as (110a) and 'INTEGLEX' as (110b).

(110) a. λT_{chart}:*RecType*

b.
$$\begin{array}{c}
\lambda T_{\text{phon}}{:}\textit{RecType}\ . \\
T_{\text{chart}} \wedge_{\text{concat}} \begin{bmatrix} \text{e}_{\text{incr}_{\text{e}}(T_{\text{chart}})}{:}T_{\text{phon}} \\ \text{e}{:}\big[\text{e}_{\text{incr}_{\text{e}}(T_{\text{chart}})}{:}\text{start}(\Uparrow^2\text{e}_{\text{incr}_{\text{e}}(T_{\text{chart}})})\big]^\frown\big[\text{e}_{\text{incr}_{\text{e}}(T_{\text{chart}})}{:}\text{end}(\Uparrow^2\text{e}_{\text{incr}_{\text{e}}(T_{\text{chart}})})\big] \end{bmatrix} \\
\text{Lex}(T_{\text{phon}}, T_{\text{sign}})\ \text{resource}_A \qquad s_{i,A} :_A T_{\text{infstate}} \qquad u^* :_A T_{\text{phon}} \\
T_{\text{infstate}} \sqsubseteq \big[\text{shared}{:}\big[\text{current-utterance}{:}T_{\text{chart}}\big]\big] \\
\hline
s_{i+1,A} :_A T_{\text{infstate}}\ \boxed{\wedge}\ \big[\text{shared}{:}\big[\text{current-utterance}{:}\text{f}_{\text{INTEGLEX}}(T_{\text{chart}}, T_{\text{phon}})\big]\big]
\end{array}$$

We now need action rules that will add signs to the chart which are derived from the lexical resources for signs associated with phonological types.

In (111) we specify an action rule INTEGPROPNAME for integrating proper names into the chart.

(111)
$$\begin{array}{c}
\text{Lex}_{\text{PropName}}(T_{\text{Phon}}, a)\ \text{resource}_A \qquad T_{\text{chart}} \sqsubseteq \begin{bmatrix} \text{e}_j{:}T_{\text{phon}} \\ \text{e}{:}\begin{bmatrix} \text{t}_k{:}[\text{e}_j{:}\text{start}(\Uparrow^2\text{e}_j)] \\ \text{t}_l{:}[\text{e}_j{:}\text{end}(\Uparrow^2\text{e}_j)] \end{bmatrix} \end{bmatrix} \\
s_{i,A} :_A T_{\text{infstate}} \\
s_{i,A}.\text{shared.current-utterance} :_A T_{\text{chart}} \\
\hline
s_{i+1,A} :_A T_{\text{infstate}}\ \boxed{\wedge} \begin{bmatrix} \text{shared}{:} \begin{bmatrix} \text{current-utterance}{:}T_{\text{chart}} \wedge \\ \begin{bmatrix} \text{e}_j{:}T_{\text{phon}} \\ \text{e}_{\text{incr}_e(T_{\text{chart}})}{:}\text{Lex}_{\text{PropName}}(T_{\text{phon}}, a) \wedge \\ \qquad [\text{s-event}{:}[\text{e}{=}\Uparrow^2\text{e}_j{:}\textit{Phon}]] \\ \text{e}{:}\begin{bmatrix} \text{t}_k{:}[\text{e}_{\text{incr}_e(T_{\text{chart}})}{:}\text{start}(\Uparrow^2\text{e}_{\text{incr}_e(T_{\text{chart}})})] \\ \text{t}_l{:}[\text{e}_{\text{incr}_e(T_{\text{chart}})}{:}\text{end}(\Uparrow^2\text{e}_{\text{incr}_e(T_{\text{chart}})})] \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}
\end{array}
\quad
\begin{array}{l}
T_{\text{chart}} \not\sqsubseteq \\
\begin{bmatrix} \text{e}_j{:}T_{\text{phon}} \\ \text{e}_n{:}\text{Lex}_{\text{PropName}}(T_{\text{phon}}, a) \wedge \\ \quad [\text{s-event}{:}[\text{e}{=}\Uparrow^2\text{e}_j{:}\textit{Phon}]] \end{bmatrix} \\
\text{for any } n
\end{array}$$

For common nouns we have a similar rule, INTEGCOMMONNOUN, as in (112).

(112)
$$\begin{array}{c}
\text{Lex}_{\text{CommonNoun}}(T_{\text{Phon}}, p)\ \text{resource}_A \qquad T_{\text{chart}} \sqsubseteq \begin{bmatrix} \text{e}_j{:}T_{\text{phon}} \\ \text{e}{:}\begin{bmatrix} \text{t}_k{:}[\text{e}_j{:}\text{start}(\Uparrow^2\text{e}_j)] \\ \text{t}_l{:}[\text{e}_j{:}\text{end}(\Uparrow^2\text{e}_j)] \end{bmatrix} \end{bmatrix} \\
s_{i,A} :_A T_{\text{infstate}} \\
s_{i,A}.\text{shared.current-utterance} :_A T_{\text{chart}} \\
\hline
s_{i+1,A} :_A T_{\text{infstate}}\ \boxed{\wedge} \begin{bmatrix} \text{shared}{:} \begin{bmatrix} \text{current-utterance}{:}T_{\text{chart}} \wedge \\ \begin{bmatrix} \text{e}_j{:}T_{\text{phon}} \\ \text{e}_{\text{incr}_e(T_{\text{chart}})}{:}\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, p) \wedge \\ \qquad [\text{s-event}{:}[\text{e}{=}\Uparrow^2\text{e}_j{:}\textit{Phon}]] \\ \text{e}{:}\begin{bmatrix} \text{t}_k{:}[\text{e}_{\text{incr}_e(T_{\text{chart}})}{:}\text{start}(\Uparrow^2\text{e}_{\text{incr}_e(T_{\text{chart}})})] \\ \text{t}_l{:}[\text{e}_{\text{incr}_e(T_{\text{chart}})}{:}\text{end}(\Uparrow^2\text{e}_{\text{incr}_e(T_{\text{chart}})})] \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}
\end{array}
\quad
\begin{array}{l}
T_{\text{chart}} \not\sqsubseteq \\
\begin{bmatrix} \text{e}_j{:}T_{\text{phon}} \\ \text{e}_n{:}\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, p) \wedge \\ \quad [\text{s-event}{:}[\text{e}{=}\Uparrow^2\text{e}_j{:}\textit{Phon}]] \end{bmatrix} \\
\text{for any } n
\end{array}$$

Finally for indefinite articles we have INTEGINDEFART given in (113).

(113)

$$\cfrac{\begin{array}{c}\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})\ \text{resource}_A \\ s_{i,A} :_A T_{\text{infstate}} \\ s_{i,A}.\text{shared.current-utterance} :_A T_{\text{chart}}\end{array} \qquad T_{\text{chart}} \sqsubseteq \begin{bmatrix}\text{e}_j{:}T_{\text{phon}} \\ \text{e}{:}\begin{bmatrix}\text{t}_k{:}\big[\text{e}_j{:}\text{start}(\Uparrow^2\text{e}_j)\big] \\ \text{t}_l{:}\big[\text{e}_j{:}\text{end}(\Uparrow^2\text{e}_j)\big]\end{bmatrix}\end{bmatrix}}{s_{i+1,A} :_A T_{\text{infstate}}\ \boxed{\dot\wedge}\ \begin{bmatrix}\text{shared:}\begin{bmatrix}\text{current-utterance:}T_{\text{chart}}\dot\wedge \\ \begin{bmatrix}\text{e}_j{:}T_{\text{phon}} \\ \text{e}_{\text{incr}_e(T_{\text{chart}})}{:}\text{Lex}_{\text{IndefArt}}(T_{\text{phon}})\ \dot\wedge \\ \qquad\big[\text{s-event:}\big[\text{e}{=}\Uparrow^2\text{e}_j{:}\mathit{Phon}\big]\big] \\ \text{e:}\begin{bmatrix}\text{t}_k{:}\big[\text{e}_{\text{incr}_e(T_{\text{chart}})}{:}\text{start}(\Uparrow^2\text{e}_{\text{incr}_e(T_{\text{chart}})})\big] \\ \text{t}_l{:}\big[\text{e}_{\text{incr}_e(T_{\text{chart}})}{:}\text{end}(\Uparrow^2\text{e}_{\text{incr}_e(T_{\text{chart}})})\big]\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}} \quad \begin{array}{l}T_{\text{chart}} \not\sqsubseteq \\ \begin{bmatrix}\text{e}_j{:}T_{\text{phon}} \\ \text{e}_n{:}\text{Lex}_{\text{IndefArt}}(T_{\text{phon}})\ \dot\wedge \\ \quad\big[\text{s-event:}\big[\text{e}{=}\Uparrow^2\text{e}_j{:}\mathit{Phon}\big]\big]\end{bmatrix} \\ \text{for any } n\end{array}$$

These three rules are identical except for the different categories involved. They are tacit action rules in that they do not require any external event such as an utterance in order for their premises (or preconditions) to be met. This means that we have to be careful to avoid them being used repeatedly adding the same information over and over to the chart. The side conditions one these rules are meant to check that the information that they will add is not already present on the chart.

Recall that we use ‘t$_i$’ where i is a natural number as the distinguished labels in records that represent strings. Thus the type (114) is the type of strings whose $k+1$th and $l+1$th positions are of type T_1 and T_2 respectively.

(114) $\begin{bmatrix}\text{t}_k & : & T_1 \\ \text{t}_l & : & T_2\end{bmatrix}$

Note that because we are modelling strings as records these types can be merged as required by these rules. Thus the merge of (114) with (115a) will be (115b).

(115) a. $\begin{bmatrix}\text{t}_k & : & T_1' \\ \text{t}_l & : & T_2'\end{bmatrix}$

b. $\begin{bmatrix}\text{t}_k & : & T_1\dot\wedge T_1' \\ \text{t}_l & : & T_2\dot\wedge T_2'\end{bmatrix}$

After integrating lexical sign types into the chart, the next step is to integrate rules from our resources that apply to strings which could begin with a lexical sign of this type. We will use an action rule called IntegRule which integrates a syntactic rule of the kind we have discussed into the chart. It is given in (116).

The action rule in (116) adds a rule to the chart if there is already something on the chart of the right type to match its “left corner”, that is the first type in the string type which is the domain

(116)

$$
\begin{array}{c}
s_{i,A} :_A T_{\text{infstate}} \qquad s_{i,A}.\text{shared.current-utterance} :_A T_{\text{chart}} \\
T_{\text{chart}} \sqsubseteq \begin{bmatrix} \mathrm{e}_k{:}T_{\text{sign}} \\ \mathrm{e}\ {:}T_{\text{evpref}}{}^\frown T_{\text{end}} \end{bmatrix} \qquad \text{incr}_e(s_{i,A}.\text{shared.current-utterance}) = n \\
T_{\text{cat}} \in \{\mathit{NP}, \mathit{VP}, \ldots\} \qquad T_{\text{sign}} \sqsubseteq T_{\text{cat}} \qquad T_{\text{end}} \sqsubseteq \big[\mathrm{e}_k{:}\text{end}(\mathrm{e}_k)\big] \\
f_{\text{rule}}\ \text{resource}_A \qquad f_{\text{rule}} : T_{\text{rule}} \\
T_{\text{rule}} = (T_{\text{leftcorner}}{}^\frown T_{\text{rest}} \rightarrow \mathit{Type}) \qquad T_{\text{sign}} \sqsubseteq T_{\text{leftcorner}} \\
\hline
s_{i+1,A} :_A T_{\text{infstate}} \boxed{\mathbin{\dot\wedge}} \\
\begin{bmatrix} \text{shared:} \begin{bmatrix} \text{current-utterance:} T_{\text{chart}} \mathbin{\dot\wedge} \begin{bmatrix} \mathrm{e}_n{:} \begin{bmatrix} \text{rule=}f_{\text{rule}}{:}T_{\text{rule}} \\ \text{fnd=}s_{i,A}.\text{shared.current-utterance.e}_k{:}T_{\text{sign}} \\ \text{req=}T_{\text{rest}}{:}\mathit{Type} \end{bmatrix} \\ \mathrm{e}{:}T_{\text{evpref}}{}^\frown (T_{\text{end}} \mathbin{\dot\wedge} \big[\mathrm{e}_n{:}\text{start}(\Uparrow^3\mathrm{e}_n)^\frown\text{end}(\Uparrow^3\mathrm{e}_n)\big]) \end{bmatrix} \end{bmatrix} \end{bmatrix}
\end{array}
\qquad
\begin{array}{l}
T_{\text{chart}} \not\sqsubseteq \\
\begin{bmatrix} \mathrm{e}_l\text{=}f_{\text{rule}}{:}T_{\text{rule}} \\ \mathrm{e}{:}T_{\text{evpref}}{}^\frown \\ \quad \big[\mathrm{e}_1{:}\text{start}(\Uparrow^3\mathrm{e}_l)^\frown\text{end}(\Uparrow^3\mathrm{e}_l)\big] \end{bmatrix} \\
\text{for any } l
\end{array}
$$

type of the function which models the rule. It marks the rule as having its left corner found and requiring the remaining string type (without the left corner) as what is still required.[14]

The final action rule that we need in order to build charts involves combining an event with a non-empty requirement with an event of a type matching the requirement whose start coincides with the end of the first event. We will call this action rule COMBINE. There are two variants of this action rule: one for the case where what is required is a string of category signs of greater than length 1 and one for the case where what is required is a string of a single category sign. In the first case we need to create a new event with a requirement which is the remainder of the requirement after removing the left corner of the original requirement and a found string which concatenates the found event at the end of the original found event string. In the second case we need to add an event of the type which results from applying the rule to the concatenation of the found event to original found event string. As we only have binary rules in our small grammar the first case will not be necessary here as we will only introduce a rule onto the chart when we have found an event matching its first element and the requirement result from this addition will thus be a string consisting of a single event of a given category type. We will thus only introduce the action rule for the second case here. We do this in (117).

[14] The fact that the new event has a non-empty requirement for future events means that it corresponds to what is known in the chart parsing literature as an active edge and the new event encodes a dotted rule.

(117)

$$\frac{s_{i,A} :_A T_{\text{infstate}} \qquad s_{i,A}.\text{shared.current-utterance} :_A T_{\text{chart}} \\ T_{\text{chart}} \sqsubseteq \begin{bmatrix} \text{e}_f{:}T_{\text{sign}_1} \\ \text{e}_k{:} \begin{bmatrix} \text{rule}{=}f_{\text{rule}}{:}(T \to \mathit{Type}) \\ \text{fnd}{=}\Uparrow\text{e}_f{:}\mathit{Sign} \\ \text{req}{=}T_{\text{sign}_2}{:}\mathit{Type} \end{bmatrix} \\ \text{e}_l{:}T_{\text{sign}_3} \\ \text{e}{:}T_1{}^\frown\begin{bmatrix}\text{e}_f{:}\text{start}(\Uparrow^3\text{e}_f)\end{bmatrix}{}^\frown T_2{}^\frown \\ \qquad \begin{bmatrix} \text{e}_f{:}\text{end}(\Uparrow^3\text{e}_f) \\ \text{e}_k{:}\text{start}(\Uparrow^3\text{e}_k){}^\frown\text{end}(\Uparrow^3\text{e}_k) \\ \text{e}_l{:}\text{start}(\Uparrow^3\text{e}_l) \end{bmatrix}{}^\frown T_3{}^\frown \\ \qquad\qquad \begin{bmatrix}\text{e}_l{:}\text{end}(\Uparrow^3\text{e}_l)\end{bmatrix}{}^\frown T_4 \end{bmatrix} \\ 1 \leq i \leq 3 \qquad T_{\text{sign}_i} \sqsubseteq T_i \qquad T_i \in \{\mathit{NP}, \mathit{VP}, \ldots\} \\ \max_e(s_{i,A}.\text{shared.current-utterance}) = n}{s_{i+1,A} :_A T_{\text{infstate}}\boxed{\wedge} \begin{bmatrix} \text{shared:} \begin{bmatrix} \text{current-utterance:}T_{\text{chart}}\wedge \begin{bmatrix} \text{e}_f{:}T_{\text{sign}_1} \\ \text{e}_k{:} \begin{bmatrix} \text{rule}{=}f_{\text{rule}}{:}(T \to \mathit{Type}) \\ \text{fnd}{=}\Uparrow\text{e}_f{:}\mathit{Sign} \\ \text{req}{=}T_{\text{sign}_2}{:}\mathit{Type} \end{bmatrix} \\ \text{e}_l{:}T_{\text{sign}_3} \\ \text{e}_{n+1}{:}r.\text{e}_k.\text{rule}(r.\text{e}_f{}^\frown r.\text{e}_l) \\ \text{e}{:}T_1{}^\frown\begin{bmatrix} \text{e}_f{:}\text{start}(\Uparrow^3\text{e}_f) \\ \text{e}_{n+1}{:}\text{start}(\Uparrow^3\text{e}_{n+1}) \end{bmatrix}{}^\frown T_2{}^\frown \begin{bmatrix} \text{e}_k{:}\text{start}(\Uparrow^3\text{e}_k){}^\frown\text{end}(\Uparrow^3\text{e}_k) \\ \text{e}_l{:}\text{start}(\Uparrow^3\text{e}_l) \end{bmatrix} \\ \qquad {}^\frown T_3{}^\frown \begin{bmatrix} \text{e}_l{:}\text{end}(\Uparrow^3\text{e}_l) \\ \text{e}_{n+1}{:}\text{end}(\Uparrow^3\text{e}_{n+1}) \end{bmatrix}{}^\frown T_4 \end{bmatrix} \end{bmatrix} \end{bmatrix}}$$

$$T_{\text{chat}} \not\sqsubseteq \begin{bmatrix} \text{e}_f{:}T_{\text{sign}_1} \\ \text{e}_l{:}T_{\text{sign}_3} \\ \text{e}_i{:}f_{\text{rule}}(\text{e}_f{}^\frown\text{e}_l) \\ \text{e}{:}\mathit{Rec}^*{}^\frown \begin{bmatrix} \text{e}_i{:}\text{start}(\Uparrow\text{e}_i) \\ \text{e}_f{:}\text{start}(\Uparrow\text{e}_f) \end{bmatrix}{}^\frown \mathit{Rec}^* \end{bmatrix}$$

for any i

3.6 Summary of resources introduced

This summary does not include the resources for chart processing introduced in Section 3.5.

3.6.1 Universal grammar resources

3.6.1.1 Types

Loc — a basic type

l : *Loc* iff l is a region in three dimensional space

Phon — a basic type

e : *Phon* iff e is a phonological event

$$\textit{SEvent} \;—\; \begin{bmatrix} \text{e-loc} & : & \textit{Loc} \\ \text{sp} & : & \textit{Ind} \\ \text{au} & : & \textit{Ind} \\ \text{e} & : & \textit{Phon} \\ \text{c}_{\text{loc}} & : & \text{loc(e,e-loc)} \\ \text{c}_{\text{sp}} & : & \text{speaker(e,sp)} \\ \text{c}_{\text{au}} & : & \text{audience(e,au)} \end{bmatrix} \text{(as in Chapter 2)}$$

Ppty — $(\begin{bmatrix}\text{x}:\textit{Ind}\end{bmatrix}\to\textit{RecType})$

Quant — $(\textit{Ppty}\to\textit{RecType})$

Cont — $\textit{RecType}\vee\textit{Ppty}\vee\textit{Quant}\vee(\textit{Ppty}\to\textit{Quant})$

Cat — a basic type

s, np, det, n, v, vp : *Cat*

$$\textit{Syn} \;—\; \begin{bmatrix} \text{cat} & : & \textit{Cat} \\ \text{daughters} & : & \textit{Sign}^* \end{bmatrix}$$

Sign — a basic type

$$\sigma : \textit{Sign} \text{ iff } \sigma : \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{syn} & : & \textit{Syn} \\ \text{cont} & : & \textit{Cont} \end{bmatrix}$$

SignType — a basic type

T : *SignType* iff $T \sqsubseteq$ *Sign* (as in Chapter 2)

$$S \;—\; \begin{bmatrix} \textit{Sign} \\ \text{syn}:\begin{bmatrix}\text{cat=s}:\textit{Cat}\end{bmatrix} \end{bmatrix}$$

NP — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=np:}\textit{Cat}\end{bmatrix}\end{bmatrix}$

Det — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=det:}\textit{Cat}\end{bmatrix}\end{bmatrix}$

N — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=n:}\textit{Cat}\end{bmatrix}\end{bmatrix}$

V — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=v:}\textit{Cat}\end{bmatrix}\end{bmatrix}$)

VP — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=vp:}\textit{Cat}\end{bmatrix}\end{bmatrix}$)

NoDaughters — $\begin{bmatrix}\text{syn:}\begin{bmatrix}\text{daughters=}\varepsilon\text{:}\textit{Sign}^*\end{bmatrix}\end{bmatrix}$

3.6.1.2 Predicates

with arity $\langle$*Phon*, *Loc*$\rangle$

loc — e : loc(u, l) iff u is located at l in e

with arity $\langle$*Phon*, *Ind*$\rangle$

speaker — e : speaker(u, a) iff u is the speaker of u in e

audience — e : audience(u, a) iff u is the audience of u in e

with arity $\langle$*Ppty*, *Ppty*$\rangle$ **New!**

exist — s : exist(P,Q) iff $[\downarrow P] \cap [\downarrow Q \upharpoonright s] \neq \emptyset$

with arity $\{\langle T\rangle \mid T$ is a type$\}$ **New!**

be — e : be(a) iff $a\varepsilon e$

3.6.1.3 Lexicon

Lex

If T_{phon} is a phonological type (that is, $T_{\text{phon}} \sqsubseteq$ *Phon*) and T_{sign} is a sign type (that is, $T_{\text{sign}} \sqsubseteq$ *Sign*), then we shall use Lex(T_{phon}, T_{sign}) to represent

(($T_{\text{sign}} \mathbin{\dot\wedge} \begin{bmatrix}\text{s-event:}\begin{bmatrix}\text{e:}T_{\text{phon}}\end{bmatrix}\end{bmatrix}$) $\dot\wedge$ *NoDaughters*)

SemCommonNoun(p)

If p is a predicate, then SemCommonNoun(p) is

$$\lambda r{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \quad : \quad p(r.\text{x})\ \big]$$

$\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, p)$

If T_{phon} is a phonological type and p is a predicate, then $\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, p)$ is

$$\text{Lex}(T_{\text{phon}}, \mathit{N})\ \underset{\cdot}{\wedge}\ \big[\text{cont=SemCommonNoun}(p)\text{:}\mathit{Ppty}\big]$$

SemPropName(a)

If a:*Ind*, then SemPropName(a) is

$$\lambda P{:}\mathit{Ppty}\ .\ P(\big[\text{x=}a\big])$$

$\text{Lex}_{\text{PropName}}(T_{\text{Phon}}, a)$

If T_{Phon} is a phonological type and a:*Ind*, then $\text{Lex}_{\text{PropName}}(T_{\text{Phon}}, a)$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{NP})\ \underset{\cdot}{\wedge}\ \big[\text{cont=SemPropName}(a)\text{:}\mathit{Quant}\big]$$

SemIndefArt

$$\lambda Q{:}\mathit{Ppty}\ .\\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix} \text{restr=}Q & : & \mathit{Ppty} \\ \text{scope=}P & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{bmatrix}$$

$\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{Det})\ \underset{\cdot}{\wedge}\ \big[\text{cont=SemIndefArt:}(\mathit{Ppty}\rightarrow\mathit{Quant})\big]$$

SemBe

$$\lambda \mathcal{Q}{:}\mathit{Quant}\ .\\ \lambda r_1{:}\big[\text{x:}\mathit{Ind}\big]\ .\\ \mathcal{Q}(\lambda r_2{:}\big[\text{x:}\mathit{Ind}\big]\ .\ \begin{bmatrix} \text{x=}r_2.\text{x},\ r_1.\text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{be(x)} \end{bmatrix})$$

$\text{Lex}_{\text{be}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{be}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{V})\ \underset{\cdot}{\wedge}\ \big[\text{cont=SemBe:}(\mathit{Quant}\rightarrow\mathit{Ppty})\big]$$

3.6.1.4 Constituent structure

RuleDaughters($T_{\text{daughters}}$, T_{mother})

If T_{mother} is a sign type and $T_{\text{daughters}}$ is a type of strings of signs then

RuleDaughters($T_{\text{daughters}}$, T_{mother})

is

$$\lambda u : T_{\text{daughters}} \,.\, T_{\text{mother}} \wedge \left[\text{syn:}\left[\text{daughters}{=}u{:}T_{\text{daughters}}\right]\right]$$

ConcatPhon

$$\lambda u{:}\left[\text{s-event:}\left[\text{e:}\textit{Phon}\right]\right]^{+} .\\ \left[\ \text{s-event} \ : \ \left[\ \text{e=concat}_i(u[i].\text{s-event.e}) \ : \ \textit{Phon}\ \right]\ \right]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$

If T_{mother} is a sign type and $T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$ are sign types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1} \ldots T_{\text{daughter}_n}$$

represents

$$\text{RuleDaughters}(T_{\text{mother}}, T_{\text{daughter}_1}{}^\frown \ldots{}^\frown T_{\text{daughter}_n}) \wedge \text{ConcatPhon}$$

ContForwardApp(T_{arg}, T_{res})

If T_{arg} and T_{res} are types, then ContForwardApp(T_{arg}, T_{res}) is

$$\lambda u{:}\left[\text{cont:}(T_{\text{arg}} \to T_{\text{res}})\right]^\frown\left[\text{cont:}T_{\text{arg}}\right] .\\ \left[\text{cont=}u[0].\text{cont}(u[1].\text{cont}){:}T_{\text{res}}\right]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \ \mid \ T'_{\text{daughter}_1}(T'_{\text{daughter}_2} : T_{\text{arg}}) : T_{\text{res}}$

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and T_{arg} and T_{res} are content types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \ \mid \ T'_{\text{daughter}_1}(T'_{\text{daughter}_2} : T_{\text{arg}}) : T_{\text{res}}$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \wedge \text{ContForwardApp}(T_{\text{arg}}, T_{\text{res}})$$

3.6.1.5 Action rules

LEXRES

$$\frac{\text{Lex}(T, C)\ \text{resource}_A \qquad u :_A T}{:_A\ (\text{Lex}(T, C) \wedge \left[\text{s-event:}\left[\text{e=}u{:}T\right]\right])}$$

3.6.2 Universal speech act resources

(as in Chapter 2)

3.6.3 Universal dialogue resources

(as in Chapter 2)

3.6.4 English resources

3.6.4.1 Basic types and predicates

(as in Chapter 2)

Basic phonological types for words

{"Dudamel", "is", "a", "conductor", "Beethoven", "composer", "Uchida", "pianist", "aha", "ok"}

Witnesses for basic types

Ind — dudamel, beethoven, uchida : *Ind*

Predicates

with arity $\langle Ind \rangle$ {conductor, composer, pianist}

3.6.4.2 Grammar

Lexical sign types

{$\text{Lex}_{\text{PropName}}$("Dudamel", d),
$\text{Lex}_{\text{PropName}}$("Beethoven", b),
$\text{Lex}_{\text{IndefArt}}$("a"),
$\text{Lex}_{\text{CommonNoun}}$("composer", composer),
$\text{Lex}_{\text{CommonNoun}}$("conductor", conductor),
Lex_{be}("is"),
Lex("ok", *S*) (as in Chapter 2),
Lex("aha", *S*) (as in Chapter 2) }

Constituent structure rule components

CnstrIsA

$$\lambda u{:}V \wedge\!\!\!\!\cdot\; \begin{bmatrix}\text{s-event:}\begin{bmatrix}\text{e:“is”}\end{bmatrix}\end{bmatrix} {}^\frown \mathit{NP} \wedge\!\!\!\!\cdot\; \begin{bmatrix}\text{syn:}\begin{bmatrix}\text{daughters:}\mathit{Det} \wedge\!\!\!\!\cdot\; \begin{bmatrix}\text{s-event:}\begin{bmatrix}\text{e:“a”}\end{bmatrix}\end{bmatrix} \\ {}^\frown N \wedge\!\!\!\!\cdot\; \begin{bmatrix}\text{cont:}\mathit{Ppty}\end{bmatrix}\end{bmatrix}\end{bmatrix}.$$

$$\mathit{VP} \wedge\!\!\!\!\cdot\; \begin{bmatrix}\text{cont=}u\text{[2].syn.daughters[2].cont:}\mathit{Ppty}\end{bmatrix}$$

Constituent structure rules

$$\begin{array}{l}\{S \longrightarrow NP\ VP \mid NP'(VP'{:}Ppty){:}RecType, \\ \ NP \longrightarrow Det\ N \mid Det'(N'{:}Ppty){:}Quant, \\ \ VP \longrightarrow V\ NP \wedge\!\!\!\!\cdot\; \text{CnstrIsA}, \\ \ VP \longrightarrow V\ NP \mid V'(NP'{:}Quant){:}Ppty\}\end{array}$$

3.7 Summary

This chapter has focussed on a view of grammar within a theory of action. We started by relating the kind of constituent structure found in syntax to the hierarchical structure of event types which seems necessary for even non-linguistic agents to interact effectively with their environment and to learn new types of actions efficiently by exploiting components of actions they already know how to perform. The kind of syntax we proposed builds closely on the insights of Head-Driven Phrase Structure Grammar (HPSG) and exploits the similarities between the feature structures used there and records. One major difference between the two approaches is that we have both records and record types where HPSG has just feature structures which sometimes leads to confusion as to whether they should be thought of intuitively as objects or types within the theory.

Another important difference is that the type theoretical approach includes functions and function types which cannot be characterized in terms of feature structures. This was important, for example, when we modelled phrase structure rules in terms of functions and associated action rules. It was, of course, of central importance when we turned our attention to semantics, where much of what we did in this chapter followed the classical formal semantic analyses proposed by Montague.

If we are going to give a theoretical account of language in terms of a theory of action, we need to take seriously the fact that language is processed incrementally, that is, not in terms of complete sentences but in terms of small phonological events corresponding to words or parts of words. Our suggestion for this was to adopt the notion of parse chart which was developed in computational linguistics in the 1980's, developed from formal methods used for the parsing of programming languages in computer science. We sketched how chart parsing could be conceived of in terms of our types and action rules and showed how this could be integrated into our view of updates carried out on information states in terms of gameboards. Thus the incremental processing of an utterance becomes part of the general theory of information state update needed to account for dialogical interaction.

While we have included both syntax and semantics in this framework and taken a fairly detailed look at how incremental parsing can be incorporated in this approach, the actual grammatical phenomena that we have looked at are linguistically trivial. In Part II we will look at a variety of linguistic phenomena and argue that the approach we have developed provides theoretically interesting insights both in terms of old problems and into new problems that arise when we view language from a dialogical perspective.

Part II

Towards a dialogical view of semantics

Chapter 4

Reference and mental states

4.1 Montague's PTQ as a semantic benchmark

In this chapter[1] and the following chapters we will extend the linguistic coverage of the toy grammar we presented in Chapter 3. We will take Montague's PTQ (Montague, 1973, 1974) as providing a benchmark of linguistic phenomena that need to be covered and try to cover a sizeable part of what Montague covered, although we will add a few things which are obviously closely related to Montague's original benchmark and which have been treated subsequently in the literature.

For many of the phenomena we discuss we will first present a treatment which is as close as possible to Montague's original treatment and then present a treatment which exploits the advantages of the approach we are proposing in this book as well as more recent developments since Montague's original work. Our aim is to show that we have something to say about all these phenomena in an overall consistent framework, that is, to show that we can cover a significant part of the benchmark using the tools we are proposing and in many cases say something new concerning a dialogical approach to these phenomena. In doing this within the space of a single book we will not be able to cover all the aspects of these phenomena which have been studied in the literature following after Montague. We hope, however, to show that it is a fruitful line of research to add a rich type theoretic perspective and a dialogical approach to current work in linguistic semantics.

4.2 Montague's treatment of proper names and a sign-based approach

The treatment of proper names that we presented in Chapter 3, encapsulated in the definition of SemPropName and $\mathrm{Lex}_{\mathrm{PropName}}$, is an adaptation of Montague's original treatment in that it

[1] An version of some of the material in this chapter has appeared in Cooper (2017b).

has the content of a proper name utterance as a quantifier generated from an individual. The essence of Montague's treatment was that if we have a proper name *Sam* whose denotation is based on an individual 'sam', then the denotation of *Sam* is the characteristic function of the set of properties possessed by the individual concept of 'sam'. Montague modelled individual concepts as functions from possible worlds to individuals. Using more or less Montague's logical notation, the denotation of *Sam* would be represented by (1).

(1) $\lambda P.P\{[\hat{}\mathrm{sam}]\}$

Here $[\hat{}\mathrm{sam}]$ represents the individual concept of 'sam', that is, that function, f, on the set of possible worlds such that for any world w, $f(w) = \mathrm{sam}$. The reason that Montague used the individual concept (and the associated special notion of application involved in applying a property to an individual concept represented by the '$\{\}$'-brackets) was to treat what is known as the Partee-puzzle concerning temperature and price which we will discuss in Chapter 5. Many subsequent researchers came to the conclusion that Montague's treatment of this puzzle was not the correct one and that the individual concept was not necessary in the treatment of proper names. Thus (1) could be simplified to (2).

(2) $\lambda P.P(\mathrm{sam})$

The content that we assigned to an utterance of *Sam* in Chapter 3 is represented in (3).

(3) $\lambda P{:}\textit{Ppty}.P(\big[\text{x=sam}\big])$

The reason that we have chosen to characterize properties as having records as their domain rather than individuals, has to do with our treatment of the Partee puzzle as we will explain in Chapter 5. Thus the reason that we have the record $\big[\text{x=sam}\big]$ as the argument to the property rather than an individual as in (4) is for the same reason as Montague introduced an individual concept.

(4) $\lambda P{:}\textit{Ppty}.P(\text{sam})$

The treatment of proper names we presented in Chapter 3 has an important advantage over Montague's original. For Montague, (1) is the result of applying an interpretation function to the linguistic expression *Sam* and a number of indices for the interpretation, $\mathfrak{A}$, a possible world, i, a time, j, and an assignment to variables, g. This is represented in (5).

(5) $[\![\ \text{Sam}\ ]\!]^{\mathfrak{A},i,j,g} = \lambda P.P\{[\hat{}\mathrm{sam}]\}$

This requires that the English expression *Sam* is always associated with the same individual 'sam' with respect to $\mathfrak{A}$ and any i, j, g related to $\mathfrak{A}$. This seems to go against the obvious fact that more than one individual can have the name *Sam*. It does not work to say that a different individual can be associated with *Sam* when it is evaluated with respect to different parameters. g is irrelevant since it is defined as an assignment to variables and the English expression *Sam* is not (associated with) a variable — it cannot be bound by a quantifier.[2] A strategy which involves varying the possible world and time to get a different individual associated with *Sam* would be defeated by the fact that there are many people called Sam in the actual world right now as well as having the unintuitive consequence that *Sam might be Sam* would be true if it is true that Sam might be somebody else called Sam and *Sam will be Sam* could be true if somebody called Sam now is somebody else called Sam at a future time. We might try saying that associating a different individual with Sam involves a different interpretation, $\mathfrak{A}'$, of the language. This has some intuitive appeal and we will discuss a variant of it in Section 4.5 in relation to a proposal by Ludlow (2014). But it will come to grief when we need to talk about two people named Sam in the same sentence unless we allow a switch in interpretation mid-sentence. While allowing interpretation to change mid-sentence may be an attractive option for other reasons it is not an option that is available on Montague's account of meaning. The normal assumption is that in cases where two individuals have the same name the language contains two expressions which are pronounced the same, for example, *Sam*$_1$ and *Sam*$_2$. This would make the treatment of proper names somewhat like Montague's treatment of pronouns in that they have silent numerical subscripts attached to them. How many *Sam*$_i$ should the language contain? One for each person named Sam, now, in the past and future and who could be named Sam in some non-actual world? If we follow the strategy with variables we would introduce countably many *Sam*$_i$ so that we would always have enough. But with assignments to variables we can always assign individuals to more that one variable without this causing a problem. But the consequence of doing this with proper names would be to say that an individual can have many names that are pronounced the same. (Sam says, "My name is Sam", not "My names are Sam".) Similarly no two individuals would have the same name, although they would be able to have distinct names which are pronounced the same. This would mean that the interpretation of *have the same name* would have to mean "have names which are pronounced the same". This might cause difficulties distinguishing between a case where we have two people named Sam and a case where people really do have distinct names which are pronounced the same such as *Ann* and *Anne* (unless you want to count this as a case of spelling the same name differently).

In contrast the analysis of proper names we presented in Chapter 3 is sign-based. It allows several sign types to share the same phonology but be associated with different contents. Treating the language in terms of signs eliminates the need for arbitrary indexing of proper names. It also allows us to individuate names in a sensible way. One way to individuate names is by the phonologies occurring in proper name sign types. Thus if we have two proper name sign types with the same phonology but contents associated with different individuals, then we have two individuals with the same name. Note that this proposal would make *Ann* and *Anne* different

[2]This claim has been called into question by later research. See Maier (2009) for discussion.

spellings of the same name since they are both associated with the same phonological type. How we individuate names can be different in different contexts if we follow the kind of proposal for counting discussed by Cooper (2011). We could, for example, introduce a field into lexical sign types for an orthographical type and allow the individuation of names by either phonology or orthography or a combination of both depending on what is most useful to the purpose at hand.

Using signs in this way seems to give us a clear, if rather simple, advantage over Montague's formal language approach, even though we have so far essentiually just transplanted Montague's analysis of proper names into our variant of a sign-based approach. However, there is a remaining question within sign-based approaches which is a kind of correlate to the need on Montague's approach to create many different names Sam_i. We are tempted to think of a "language" as being defined as a collection of sign types. Thus a person who knows English will know sign types which pair the phonological type "Sam" with various individuals who are called Sam. The problem with this is that different speakers of English will know different people named Sam and thus technically we would have to say that they speak different languages. This may well be a coherent technical notion of language. In the terminology of Chapter 3 we would say that the two agents indeed have different linguistic resources available to them. But there is also a resource which the two agents share, even if they do not have any overlap in the people named Sam that they are aware of. This is the knowledge that *Sam* is a proper name in English and can be used to name individuals. Arguably it is this knowledge which is constitutive of English, rather than the knowledge of who is actually called Sam, important though that might be for performing adequately in linguistic situations. In Chapter 3 we introduced sign type contruction operations and in particular '$\text{Lex}_{\text{PropName}}$' which maps a phonological type and an individual to an appropiate proper name sign type. We called this a universal resource since it represents the general knowledge that utterances can be used to name individuals. In the English resources we defined there we named sign types such as '$\text{Lex}_{\text{PropName}}$("Sam", sam)', where we specify both the phonological type and the individual associated with it. But, given the power of functional abstraction, we can identify (6) as an English resource where the phonological type is specified but not the particular individual.

(6) $\lambda x{:}\mathit{Ind}\ .\ \text{Lex}_{\text{PropName}}(\text{“Sam”}, x)$

Saying that an agent has this function available as an English resource could be argued to encode the fact that the agent has the knowledge that *Sam* is a proper name in English. An agent who has this resource has a recipe for constructing an appropriate sign type in their resources whenever they meet somebody called Sam. Knowing that *Sam* is a proper name in English is not a matter of knowing who is called Sam but rather a matter of knowing what to do linguistically when you encounter somebody called Sam. Thus while we have so far just taken over Montague's original analysis of proper names we have given ourselves the opportunity to recast it in terms of a theory which enables agents to update their linguistic resources as they become aware of new facts about the world.

4.3 Proper names and communication

However, what we have done so far tells us little about the communicative processes associated with utterances of proper names. In Cooper (2013b) we pointed out that this kind of analysis does not give us any way of placing the requirement on the interlocutor's gameboard that there already be a person named Sam available in order to integrate the new information onto the gameboard. As Ginzburg (2012) points out, the successful use of a proper name to refer to an individual a requires that the name be publically known as a name for a. We will follow the analysis of Cooper (2013b) in parametrizing the content. A *parametric content* is a function which maps a context to a content. As such it relates to Montague's technical notion of *meaning* in his paper 'Universal Grammar' (Montague, 1970, 1974) where he regarded meaning as a function from possible worlds and contexts of use to denotations.[3] This also corresponds to the notion of *character* in Kaplan (1978).

We will take a context to be a situation modelled as a record. A simple proposal for a parametric content for a proper name might be (7).

(7) $$\lambda c{:}\big[\text{x:}\textit{Ind}\big]\ .\ \lambda P{:}\textit{Ppty}\ .\ P(c)$$

This would allow any record with an individual labelled 'x' to be mapped to a proper name content. Recall that the label 'x' is picked up by the notion of property that we defined in Chapter 3 as being of type ($\big[\text{x:}\textit{Ind}\big]\rightarrow\textit{RecType}$), an example being (8).

(8) $$\lambda r{:}\big[\text{x:}\textit{Ind}\big]\ .\ \big[\text{e:run}(r.\text{x})\big]$$

Associating the phonological type "Sam" with (7) would essentially be a way of encapsulating in the interpretation of *Sam* what is expressed by (6) — namely, that potentially any individual can be called Sam. However, we want the parametric content of *Sam* to be more restrictive than this. It is going to be the tool that we use to help us identify an appropriate referent when we are confronted with an utterance of type "Sam". The obvious constraint that we should place is that the referent is indeed named Sam. Thus we can restrict (7) so that it is an appropriate parametric content for *Sam* rather than something that appears to be a parametric content appropriate to proper names in general. The modification is given in (9).

(9) $$\lambda c{:}\begin{bmatrix}\text{x:}\textit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(c)$$

[3] See Section 4 (Semantics: Theory of Reference) of 'Universal Grammar'.

We will see in Section 4.6 that there is more that one component in the context since we will use a separate component to assign individuals to unbound pronouns. In preparation for this we will use the label '$\mathfrak{c}$' for the component of the context we are introducing here as shown in (10).

(10) $\lambda c{:}\left[\mathfrak{c}{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\right]$.
$\lambda P{:}\mathit{Ppty}\ .\ P(c.\mathfrak{c})$

In order to ensure the presence of the label '$\mathfrak{c}$' in contexts we will define a type of contexts, *Cntxt*. We will refine the characterization of this type as we progress. For now it is characterized as in (11).

(11) *Cntxt* designates

$\left[\ \mathfrak{c}\ :\ \mathit{Rec}\ \right]$

This treatment of proper names is closely related to treatments that were proposed earlier in situation semantics (Gawron and Peters, 1990; Cooper, 1991; Barwise and Cooper, 1993). A more recent close relation is Maier's (2009) proposal for the treatment of proper names in terms of layered discourse representation theory (LDRT). Maier points out in a useful overview of the history of semantic treatments of proper names that this view of proper names is a hybrid of the descriptivist and referential approaches: it uses a description like "named Sam" to provide a presuppositional restriction on the kind of referent which can be assigned to the proper name. (11) maps a context in which there is an individual named Sam to a proper name content based on that individual. Care has to be taken with the predicate 'named' on this kind of analysis. It is important that it not be too restrictive, for example, requiring the legal registering of the name. It may be sufficient that someone at some point has called the individual by the name. The exact conditions under which a situation may be of a type constructed with this predicate will vary depending on the needs associated with the conversation at hand. We may, for example, take a stricter view of what it means to have a certain name if we are talking in a court of law than if we are trying to attract somebody's attention to avoid an accident on a mountainside. This flexibility of meaning "in flux" has been discussed in Cooper and Kempson (2008); Cooper (2012b); Ludlow (2014); Ginzburg and Cooper (2014); Kracht and Klein (2014) among many other places and we will return to it several times in the following chapters.

An alternative to the use of parametric contents is to use parametric signs. This could be formulated as in (12) where $\mathrm{Lex}_{\mathrm{PropName}}$ is the function for associating lexical content with phonological types that was introduced in Chapter 3.

(12) $\lambda c{:}\left[\mathfrak{c}{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\right]$.

$$\text{Lex}_{\text{PropName}}(\text{“Sam”}, c.\mathfrak{c}.\text{x})$$

Intuitively, (12) says that given a situation in which there is an individual named by the phonological type “Sam” we can construct a sign type in which the phonological type “Sam” is associated with that individual. From the point of view of the formal semantics tradition (12) is a much more radical proposal than (10). The function (10) is a close relative of Montague’s *meaning* and Kaplan’s *character*. It is a function from contexts to contents, although our theory of what contexts and contents are differs from both Montague’s and Kaplan’s proposals. The function in (12), however, is something that creates a kind of linguistic resource on the basis of a context. That is, given a context in which ‘sam’ is named by “Sam” we derive the information that linguistic signs can be used which associate “Sam” with ‘sam’. If we did not know this before we are extending the collection of linguistic resources we have available. We suspect that both parametric contents and parametric sign types could be of importance for a theory of linguistic interpretation and learning. For now, we will work with the less radical notion of parametric content.

Parametric contents as we have presented them so far are problematic for compositional semantics because the domain type of the function (representing the “presupposition”) which is the parametric content varies from case to case depending on what the intuitive presupposition of the phrase is. According to our rules it will always be some subtype of *RecType* (since we are thinking of contexts as records/situations) and we can therefore assign them to the type of partial functions from records to quantifiers, $(Rec \rightharpoonup Quant)$. However, while types of partial functions as we have defined them are useful for gathering together functions of different types into a single class they are not useful in a setting where we need to guarantee that a given function is provided with an argument which is in its domain.

For this reason we will say that a parametric content is a pair (construed as a record with two fields) containing a type and a function whose domain type is that type. We can create such a parametric content by using a redefined version of ‘SemPropName’ which we introduced in Chapter 3. Whereas the version from Chapter 3 took an individual as argument and created the content of a name of that individual, the new version will take a phonological type as argument and create a parametric content requiring an individual named by that phonological type. The new version is given in (13).

(13) SemPropName(T), where T is a phonological type, is

$$\left[\begin{array}{lll} \text{bg} & = & \left[\mathfrak{c}:\left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{e}:\text{named}(\text{x}, T)\end{array}\right]\right] \\ \text{fg} & = & \lambda c{:}\ \left[\mathfrak{c}:\left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{e}:\text{named}(\text{x}, T)\end{array}\right]\right] . \\ & & \qquad\qquad \lambda P{:}\mathit{Ppty}\ .\ P(c.\mathfrak{c}) \end{array}\right]$$

Here the field labelled 'bg' ("background") contains a record type and the field labelled 'fg' ("foreground") is a function whose domain type is the background record type. From now on we will mean records of this kind by *parametric content*.

While it is useful to think of parametric contents as records of this kind the record notation in (13) is clumsy and repetitive and we will need to talk a good deal about different parametric contents. We will therefore use a more concise notation for such records as given in (14).

(14) $\ulcorner\lambda v\!:\!T\;.\;\varphi\urcorner$ represents the record

$$\begin{bmatrix} \text{bg} & = & T \\ \text{fg} & = & \lambda v\!:\!T\;.\;\varphi \end{bmatrix}$$

This means that we can write (13) more economically as (15)

(15) SemPropName(T), where T is a phonological type, is

$$\ulcorner\lambda c\text{:}\ \begin{bmatrix}\mathfrak{c}\text{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, }T\text{)}\end{bmatrix}\end{bmatrix}\;.\;\lambda P\text{:}\mathit{Ppty}\;.\;P(c.\mathfrak{c})\urcorner$$

The type of a parametric content of proper names is (16) where we use *CntxtType* to be the type of types which are subtypes of *Cntxt*, that is T:*CntxtType* iff $T \sqsubseteq$ *Cntxt*.

(16) $$\begin{bmatrix} \text{bg} & : & \mathit{CntxtType} \\ \text{fg} & : & (\text{bg}{\rightarrow}\mathit{Quant}) \end{bmatrix}$$

That is, the foreground is a function from records of the background type (modelling contexts) to quantifiers. We will refer to this type as *PQuant* ("parametric quantifiers"). The universal resource $\text{Lex}_{\text{PropName}}$ for associating proper name content with phonological types, creating a sign type for a proper name, will now be redefined so that it only takes a phonological type as argument as in (17).

(17) $\text{Lex}_{\text{PropName}}(T_{\text{Phon}})$, where T_{Phon} is a phonological type,
is defined as
$\text{Lex}(T_{\text{Phon}}, \mathit{NP}) \mathbin{\dot{\wedge}} \left[\text{cnt=SemPropName}(T_{\text{Phon}})\text{:}\mathit{PQuant}\right]$

Note that the phonological type plays a dual role here. It figures once as determining the phonology of the sign and again as determining the presupposition associated with the parametric content.

There are two main questions that need to be answered about parametric contents. One concerns how the compositional semantics works and the other concerns the nature of contexts and how you compute with them. We will take the compositionality issue first. Let us assume that all signs provide us with a parametric content rather than a content. In those cases where there is no constraint on what the context must be we will use a trivial parametric content, that is, one that maps any context (modelled as a record) to the same content. Thus, for example, if we wish to represent a theory in which the intransitive verb *leave* does not place any restrictions on the context, we could represent its parametric content as (18a) which is of the type for parametric properties (*PPpty*) given in (18b).

(18) a. $\ulcorner\lambda c{:}\mathit{Cntxt}.\lambda r{:}\big[\mathrm{x}{:}\mathit{Ind}\big].\ \big[\mathrm{e}{:}\mathrm{leave}(r.\mathrm{x})\big]\urcorner$

b. $\begin{bmatrix} \mathrm{bg} & : & \mathit{CntxtType} \\ \mathrm{fg} & : & (\mathrm{bg}\to\mathit{Ppty}) \end{bmatrix}$

The foreground of this parametric property will map any context c to the function (19) which does not depend in any way on c.

(19) $\lambda r{:}\big[\mathrm{x}{:}\mathit{Ind}\big].\ \big[\mathrm{e}{:}\mathrm{leave}(r.\mathrm{x})\big]$

Such a content could be introduced by a resource for lexical content construction 'SemIntransVerb' as characterized in (20), where T_{bg}, the "background" or "presupposition" type, is a record type and p is a predicate with arity $\langle \mathit{Ind}\rangle$.

(20) SemIntransVerb(T_{bg}, p) is

$$\ulcorner\lambda c{:}T_{\mathrm{bg}}\ .\ \lambda r{:}\big[\mathrm{x}{:}\mathit{Ind}\big]\ .\ \big[\ \mathrm{e}\ :\ p(r.\mathrm{x})\ \big]\urcorner$$

Note that if (20) is the only way of constructing parametric content for lexical intransitive verbs, then although it is possible to place restrictions on the context by choosing a non-trivial record type (something other than *Rec*) for T_{bg} this will not have any effect on the property returned as the content. As we are not here concerned with presuppositions introduced by lexical intransitive verbs we will leave open whether it is necessary to change this. 'SemIntransVerb' will be used by the universal resource '$\mathrm{Lex}_{\mathrm{IntransVerb}}$' defined in (21), where T_{phon} is a phonological type and p is a predicate with arity $\langle \mathit{Ind}\rangle$.

(21) $\mathrm{Lex}_{\mathrm{IntransVerb}}(T_{\mathrm{phon}}, T_{\mathrm{bg}}, p)$
is defined as
$\mathrm{Lex}(T_{\mathrm{phon}}, V_i) \wedge\!\!\!\!\cdot\ \big[\mathrm{cnt}{=}\mathrm{SemIntransVerb}(T_{\mathrm{bg}}, p){:}\mathit{PPpty}\big]$

This means that the English resource corresponding to the lexical entry for *leave* can be defined as (22).

(22) $\mathrm{Lex}_{\mathrm{IntransVerb}}$(“leave”, *Rec*, leave)

A standard strategy for dealing with compositional semantics when using parametric contents is to use a version of what is known in combinatorial logic as the S-combinator. In its λ-calculus version this is (23).

(23) $\lambda z \,.\, \alpha(z)(\beta(z))$

Our version of the S-combinator including different type requirements on the context arising from the function and the argument will be (24), a preliminary version which we will modify in order to include the treatment of free pronouns in Section 4.6.

(24) If $\alpha : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow(T_1 \rightarrow T_2))\end{bmatrix}$ and $\beta : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow T_1)\end{bmatrix}$ then the *combination of* α *and* β *based on functional application*, $\alpha@\beta$, is

$$\ulcorner\lambda c{:}[\alpha.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}} \mathbin{\dot\wedge} [\beta.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}} \,.\, [\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}}(c)([\beta]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}(c))\urcorner$$

The notation $[o]_{\pi_1\rightarrow\pi_2}$ used here refers to an object like o except that any occurrence of the path π_1 has been replaced by the path π_2.

If T is a record type, the notation $[T]_{\pi_1\rightsquigarrow\pi_2}$ refers to the type like T except that the path π_1 has been replaced by π_2 provided that $\pi_1 \rightsquigarrow \pi_2$ represents an appropriate relabelling for T as characterized below on pp. 177*ff*. Otherwise $[T]_{\pi_1\rightsquigarrow\pi_2}$ is T.

If f is a function of the form $\lambda v{:}T \,.\, \varphi(\!(v)\!)$ then we use $[f]_{\pi_1\rightsquigarrow\pi_2}$ to represent (25).

(25) $\lambda v{:}[T]_{\pi_1\rightsquigarrow\pi_2} \,.\, [\varphi(\!(v)\!)]_{v.\pi_1/v.\pi_2}$

where $[\varphi(\!(v)\!)]_{v.\pi_1/v.\pi_2}$ is exactly like $\varphi(\!(v)\!)$ except that any instance of $v.\pi_1$ is replaced by $v.\pi_2$.

If r is a record of the type (26)

(26) $\begin{bmatrix} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow T) \end{bmatrix}$

for some type T and has exactly two fields, we use $[r]_{\pi_1 \rightsquigarrow \pi_2}$ to represent the record in (27)

(27) $\begin{bmatrix} \text{bg} & = & [r.\text{bg}]_{\pi_1 \rightsquigarrow \pi_2} \\ \text{fg} & = & [r.\text{fg}]_{\pi_1 \rightsquigarrow \pi_2} \end{bmatrix}$

Otherwise $[o]_{\pi_1 \rightsquigarrow \pi_2}$ refers to o.

We will discuss such relabellings in detail on pp. 177*ff*. The use of relabelling here means that in the background for the result we have kept the backgrounds of α and β separated in their own fields labelled ‘f’ (“function”) and ‘a’ (“argument”). This means that we avoid an unwanted clash of labels if α.bg and β.bg should happen to share labels.

This new method of combination for parametric contents means that we also have to adjust the sign combination operation ContForwardApp (“forward application of contents”) used in the definition of interpreted phrase structure rules (see Chapter 3, example (39)). The new version using ‘@’ rather than straightforward application is given in (28).

(28) $\lambda T_1{:}\textit{Type}\ \lambda T_2{:}\textit{Type}\ .$
$$\lambda u{:}\left[\text{cont:}\begin{bmatrix}\text{bg:}\textit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow(T_2 \rightarrow T_1))\end{bmatrix}\right]^\frown\left[\text{cont:}\begin{bmatrix}\text{bg:}\textit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow T_2)\end{bmatrix}\right]\ .$$
$$\left[\text{cont=}u[0].\text{cont@}u[1].\text{cont:}\begin{bmatrix}\text{bg:}\textit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow T_1)\end{bmatrix}\right]$$

We can use (28) to combine the contents (10) and (18). The result is given in (29) where we can show by successive applications of β-reduction that (29a–d) are all identical.

(29) a. $\ulcorner\lambda c{:}\left[\mathfrak{c}{:}\begin{bmatrix}\text{f:}\begin{bmatrix}\text{x:}\textit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\\ \text{a:}\textit{Rec}\end{bmatrix}\right]\ .$
$$(\lambda c{:}\left[\mathfrak{c}{:}\begin{bmatrix}\text{x:}\textit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\right]\ .\ \lambda P{:}\textit{Ppty}\ .\ P(c.\mathfrak{c}))(c.\mathfrak{c}.\text{f})$$
$$((\lambda c{:}\textit{Rec}\ .\ \lambda r{:}\left[\text{x:}\textit{Ind}\right]\ .\ \left[\text{e:leave}(r.\text{x})\right])(c.\mathfrak{c}.\text{a}))^\urcorner$$

b. $$\ulcorner\lambda c{:}\begin{bmatrix}\mathfrak{c}{:}\begin{bmatrix}\text{f:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\\ \text{a:}\mathit{Rec}\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\mathit{Ppty}\ .\ P(c.\mathfrak{c}.\text{f})\ (\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix}\ .\ \begin{bmatrix}\text{e:leave}(r.\text{x})\end{bmatrix})\urcorner$$

c. $$\ulcorner\lambda c{:}\begin{bmatrix}\mathfrak{c}{:}\begin{bmatrix}\text{f:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\\ \text{a:}\mathit{Rec}\end{bmatrix}\end{bmatrix}\ .\ \lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix}.\ \begin{bmatrix}\text{e:leave}(r.\text{x})\end{bmatrix}(c.\mathfrak{c}.\text{f})\urcorner$$

d. $$\ulcorner\lambda c{:}\begin{bmatrix}\mathfrak{c}{:}\begin{bmatrix}\text{f:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\\ \text{a:}\mathit{Rec}\end{bmatrix}\end{bmatrix}\ .\ \begin{bmatrix}\text{e:leave}(c.\mathfrak{c}.\text{f.x})\end{bmatrix}\urcorner$$

(29) represents the parametric content of *Sam leaves*. Given a situation containing an individual, a, named by “Sam” the function which is its foreground returns a type of situation in which a leaves. As usual this type can play the role of a “proposition”. It is “true” if there is a situation of the type and “false” if there is no situation of the type.

The background of the parametric content, that is the domain type of its foreground, is to be thought of as placing a constraint on the context. The idea is that you can only get to the non-parametric content if you have an appropriate situation available. The background of the parametric content is a type which represents a kind of *presupposition*. We shall treat presuppositions as constraints on the resources available to dialogue participants. In Chapter 2 we introduced the notion of a dialogue gameboard as a type of dialogue information state. The most obvious place to look for the referent of an utterance of a proper name is in the shared commitments represented on the gameboard representing what has been committed to in the dialogue so far. If an individual named Sam has already been introduced in the dialogue, then a subsequent utterance of *Sam* in that dialogue is most likely to refer to that individual unless there is an explicit indication to the contrary. The shared commitments on an agent's dialogue gameboard represent information that is particularly *salient* to the agent. The notion of salience in semantics was first introduced by Lewis (1979b) in connection with the analysis of definite descriptions. As Lewis says, “There are various ways for something to gain salience. Some have to do with the course of conversation, others do not.” We wish to suggest that a way of gaining salience in a conversation is by figuring in the shared commitments on the gameboard. (Ginzburg, 2012, argues that being on shared commitments, or FACTS in his terminology, is not always sufficient to indicate salience.)

A reasonable strategy, then, is to look at the shared commitments on the dialogue gameboard first and then look elsewhere if that fails. We will first explore what we need to do to match the background type of a parametric content against the type which models the shared commitments of the dialogue and then we will discuss what needs to be done if there is not a successful match with the shared commitments. In Chapter 2 we treated the gameboard as a record type. In Chapter 2, example (131), for instance, the shared commitments were represented as the type (30).

(30)
$$\left[\begin{array}{ll}\text{prev:} & \left[\begin{array}{ll}\text{prev:} & \left[\begin{array}{l}\text{prev:}\mathit{Rec}\\ \text{e:conductor(dudamel)}\end{array}\right]\\ \multicolumn{2}{l}{\text{e:composer(beethoven)}}\end{array}\right]\\ \multicolumn{2}{l}{\text{e:pianist(uchida)}}\end{array}\right]$$

Recall that with each successive updating of the shared commitments the record type representing the previous state of shared commitments was embedded under the label ‘prev’ (“previous”). This prevented label clash and also kept a record of the order in which information was introduced. As Lewis (1979b) observed, information introduced later in the dialogue tends to be more salient than information introduced earlier. Thus keeping track of the order also gives us one measure of relative salience.

In Chapter 2 we were using the Montague treatment of proper names that did not introduce the naming predicate. In this chapter we will work towards shared commitments where the naming associated with proper names is made explicit, as in (31).

(31)
$$\left[\begin{array}{ll}\text{prev:} & \left[\begin{array}{ll}\text{prev:} & \left[\begin{array}{ll}\multicolumn{2}{l}{\text{prev:}\mathit{Rec}}\\ \text{bg:} & \left[\begin{array}{l}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Dudamel”)}\end{array}\right]\\ \text{fg:} & \left[\text{e:conductor(}\Uparrow\text{bg.x)}\right]\end{array}\right]\\ \text{bg:} & \left[\begin{array}{l}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Beethoven”)}\end{array}\right]\\ \text{fg:} & \left[\text{e:composer(}\Uparrow\text{bg.x)}\right]\end{array}\right]\\ \text{bg:} & \left[\begin{array}{l}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Uchida”)}\end{array}\right]\\ \text{fg:} & \left[\text{e:pianist(}\Uparrow\text{bg.x)}\right]\end{array}\right]$$

Here we are using the label ‘bg’ to represent background information in the manner suggested by Larsson (2010) and we see also that this labelling corresponds to our use of ‘bg’ and ‘fg’ in parametric contents. Note that in this version of the shared commitments we have lost the connection with the actual individuals ‘dudamel’, ‘beethoven’ and ‘uchida’. This can be seen as an advantage if we are representing the information state of an agent in the kind of situation described in Chapter 2. If we simply inform an agent with no previous knowledge of Dudamel that Dudamel is a conductor, then the information that this agent will get is that there is somebody named Dudamel who is a conductor. There will be no connection to a particular individual of whom the agent is aware. If this is not the case, we can reinstate the connection to the individuals by using manifest fields to anchor the information as in (32).

$$(32)\quad \left[\begin{array}{ll} \text{prev:} & \left[\begin{array}{ll} \text{prev:} & \left[\begin{array}{ll} \multicolumn{2}{l}{\text{prev:}\textit{Rec}} \\ \text{bg:} & \left[\begin{array}{l} \text{x=dudamel:}\textit{Ind} \\ \text{e:named(x, ``Dudamel'')} \end{array}\right] \\ \text{fg:} & \left[\text{e:conductor}(\Uparrow\text{bg.x})\right] \end{array}\right] \\ \text{bg:} & \left[\begin{array}{l} \text{x=beethoven:}\textit{Ind} \\ \text{e:named(x, ``Beethoven'')} \end{array}\right] \\ \text{fg:} & \left[\text{e:composer}(\Uparrow\text{bg.x})\right] \end{array}\right] \\ \text{bg:} & \left[\begin{array}{l} \text{x=uchida:}\textit{Ind} \\ \text{e:named(x, ``Uchida'')} \end{array}\right] \\ \text{fg:} & \left[\text{e:pianist}(\Uparrow\text{bg.x})\right] \end{array}\right]$$

The ‘bg’-fields in (31) can be thought of as corresponding to the internal anchors of Kamp (1990); Kamp *et al.* (2011). The use of manifest fields in (32) would then correspond to the association of what they call external anchors with those internal anchors.

The task we have before us is to try to match the domain type of the function in (29), that is, the type which is the background of the paramertric content, repeated in (33), against the types of shared commitments in (31) or (32).

$$(33)\quad \left[\begin{array}{ll} \mathfrak{c}\text{:} & \left[\begin{array}{ll} \text{f:} & \left[\begin{array}{l} \text{x:}\textit{Ind} \\ \text{e:named(x, ``Sam'')} \end{array}\right] \\ \multicolumn{2}{l}{\text{a:}\textit{Rec}} \end{array}\right] \end{array}\right]$$

Intuitively, this attempt at matching should fail since there is no commitment to an individual named Sam in the shared commitments. Suppose now that we add to (31) as in (34).

$$(34)\quad \left[\begin{array}{ll} \text{prev:} & \left[\begin{array}{ll} \text{prev:} & \left[\begin{array}{ll} \text{prev:} & \left[\begin{array}{ll} \multicolumn{2}{l}{\text{prev:}\textit{Rec}} \\ \text{bg:} & \left[\begin{array}{l} \text{x:}\textit{Ind} \\ \text{e:named(x, ``Dudamel'')} \end{array}\right] \\ \text{fg:} & \left[\text{e:conductor}(\Uparrow\text{bg.x})\right] \end{array}\right] \\ \text{bg:} & \left[\begin{array}{l} \text{x:}\textit{Ind} \\ \text{e:named(x, ``Beethoven'')} \end{array}\right] \\ \text{fg:} & \left[\text{e:composer}(\Uparrow\text{bg.x})\right] \end{array}\right] \\ \text{bg:} & \left[\begin{array}{l} \text{x:}\textit{Ind} \\ \text{e:named(x, ``Uchida'')} \end{array}\right] \\ \text{fg:} & \left[\text{e:pianist}(\Uparrow\text{bg.x})\right] \end{array}\right] \\ \text{bg:} & \left[\begin{array}{l} \text{x:}\textit{Ind} \\ \text{e:named(x, ``Sam'')} \end{array}\right] \\ \text{fg:} & \left[\text{e:singer}(\Uparrow\text{bg.x})\right] \end{array}\right]$$

Intuitively, this should enable a match since this does commit to an individual named Sam. However, there is not a direct formal relationship between (33) and (34) corresponding to this intuition. We will use relabelling of record types in order to capture the relationship.

In order to match (33) against (34) we look for a relabelling, η, of (33) that would make (34) be a subtype of (33). Such a relabelling is given in (35a) which we will write as (35b).and the result of applying it to (33) is given in (35c).

(35) a. η is a function with domain $\{\mathfrak{c}.\text{f.x},\mathfrak{c}.\text{f.e},\mathfrak{c}.\text{a}\}$ such that

$$\eta(\mathfrak{c}.\text{f.x}) = \text{bg.x}$$
$$\eta(\mathfrak{c}.\text{f.e}) = \text{bg.e}$$
$$\eta(\mathfrak{c}.\text{a}) = \text{prev}^4$$

where prev^4 stands for prev.prev.prev.prev

b. $\mathfrak{c}.\text{f.x} \rightsquigarrow \text{bg.x}$
$\mathfrak{c}.\text{f.e} \rightsquigarrow \text{bg.e}$
$\mathfrak{c}.\text{a} \rightsquigarrow \text{prev}^4$

c. $$\begin{bmatrix} \text{bg} & : & \begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Sam”)} \end{bmatrix} \\ \text{prev} & : & \begin{bmatrix} \text{prev} & : & \begin{bmatrix} \text{prev} & : & \begin{bmatrix} \text{prev} & : & \textit{Rec} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

This means, then, that any situation which is of the type required by the shared commitments would, modulo the relabelling, be of the type which is the background of the parametric content under consideration, spelled out in (36).

(36) $$\ulcorner\lambda c{:}\begin{bmatrix} \mathfrak{c}{:}\begin{bmatrix} \text{f:}\begin{bmatrix} \text{x:}\textit{Ind} \\ \text{e:named(x, “Sam”)} \end{bmatrix} \\ \text{a:}\textit{Rec} \end{bmatrix} \end{bmatrix} \,.\, \begin{bmatrix} \text{e:leave}(c.\mathfrak{c}.\text{f.x}) \end{bmatrix}\urcorner$$

The background of the parametric content is being used as a presupposition which is being matched against the hearer’s current information state.

In order to give a more precise characterization of relabelling we will need two auxiliary notions. If T is a record type, we will use $\mathrm{dpaths}(T)$ to represent the dependent paths of T, that is, the set of paths which occur in any field within T in a dependent field, that is any path π that occurs in Π in some dependent field $[\ell : \langle f, \Pi\rangle]$ anywhere within T. We also use a notion of *initial subpath*. To characterize this we will use the notation given in (37).

(37) If π_1 is $\ell_1. \ \ldots \ .\ell_n$ and π_2 is $\ell_{n+1}. \ \ldots \ .\ell_m$ (where all ℓ_i are labels), then $\pi_1.\pi_2$ represents $\ell_1. \ \ldots \ .\ell_n.\ell_{n+1}. \ \ldots \ .\ell_m$

We can now characterize the notion of initial subpath as in (38) (repeated in Appendix 1).

(38) a. π_1 is an *initial subpath* of π_2, $\pi_1 \leq \pi_2$, just in case either $\pi_1 = \pi_2$ or there is some π such that $\pi_2 = \pi_1.\pi$.

b. π_1 is a *proper initial subpath* of π_2, $\pi_1 < \pi_2$, just in case there is some π such that $\pi_2 = \pi_1.\pi$.

If $\mathcal{L}$ is a set of labels, we use $\mathcal{L}^\pi$ to represent the set of potential paths $\ell_1. \ \ldots \ .\ell_n$ where n is a natural number and each ℓ_i is a member of $\mathcal{L}$. The notion of a relabelling of a record type can now be characterized as in (39).

(39) A *relabelling, η, of a record type T relative to a set of labels $\mathcal{L}$ on which T is based* is a one-one function whose domain is included in $\mathrm{paths(T)}$ and whose range is included in $\mathcal{L}^\pi$ such that

1. if $\pi_1, \pi_2 \in \mathrm{dom}(\eta)$ then $\pi_1 < \pi_2$ iff $\eta(\pi_1) < \eta(\pi_2)$
2. if $\pi \in \mathrm{dpaths}(T)$, then $\pi \in \mathrm{dom}(\eta)$
3. if $\pi \in \mathrm{paths}(T) - \mathrm{dom}(\eta)$, then it is not the case for any $\pi' \in \mathrm{dom}(\eta)$ that $\eta(\pi') = \pi$, unless $\pi' < \eta(\pi')$

Note that we allow $\eta(\pi) = \pi$. Thus while clause 2 of (39) requires each path mentioned in a dependent field of the record type (i.e. $\mathrm{dpaths}(T)$) is in the domain of the relabelling, we do not require that the relabelling actually change that path. It is, however, important that we guarantee that there is a path in the relabelled type which the dependent field can refer to. This becomes important when we have paths in dependent fields which are not total. Consider (40) which could intuitively be a type corresponding to *Sam saw a dog run*.

(40) $$\left[\begin{array}{lll} \mathrm{s} & : & \left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c} & : & \mathrm{dog(x)} \\ \mathrm{e} & : & \mathrm{run(x)} \end{array}\right] \\ \mathrm{e} & : & \mathrm{see(sam,s)} \end{array}\right]$$

Suppose that we tried to apply the relabelling in (41) to this.

(41) s.x $\rightsquigarrow$ x
s.c $\rightsquigarrow$ c
s.e $\rightsquigarrow$ e_1

We might expect this relabelling to result in the type in (42) which is ill-formed because the 'e'-field depends on a path 's' which is not present.

(42)
$$\begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c} & : & \mathrm{dog(x)} \\ \mathrm{e}_1 & : & \mathrm{run(x)} \\ \mathrm{e} & : & \mathrm{see(sam,s)} \end{bmatrix}$$

This relabelling is prevented by the requirement in clause 2 that all paths mentioned in dependent fields (the "dpaths") be assigned something by the relabelling. In this example the dpaths are 's' and 's.x' (represented just by 'x' in the abbreviatory notation). Furthermore, it is required that $\eta(\mathrm{s}) < \eta(\mathrm{s.x})$ by clause 1. That is, in intuitive terms, we must provide a path to what Sam sees in the relabelled type. In (43) is an example of a relabelling which respects the requirements.

(43) s $\rightsquigarrow$ sit
s.x $\rightsquigarrow$ sit.id.x
s.c $\rightsquigarrow$ sit.id.e
s.e $\rightsquigarrow$ sit.e

This results in (44).

(44)
$$\begin{bmatrix} \mathrm{sit} & : & \begin{bmatrix} \mathrm{id} & : & \begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & \mathrm{dog(x)} \end{bmatrix} \\ \mathrm{e} & : & \mathrm{run(x)} \end{bmatrix} \\ \mathrm{e} & : & \mathrm{see(sam,sit)} \end{bmatrix}$$

We represent the result of relabelling a type T with a relabelling, η, for T, as $[T]_\eta$. We characterize this in terms of the unique identifier notation for record types introduced in Chapter 2, p. 79. We will use three additional notions based on this notation as given in (45).

(45) a. A *unique identifier segment* is a path $\ell_1. \ldots .\ell_{n\boxed{i}}$ such that only $\ell_{n\boxed{i}}$ has a unique identifier subscript in $\ell_1, \ldots, \ell_{n\boxed{i}}$

b. A *unique identifier segmentation* is a path $\pi_{1\boxed{i}}.\pi_{2\boxed{j}}.\ \ldots\ .\pi_{n\boxed{k}}$ or $\pi_{1\boxed{i}}.\pi_{2\boxed{j}}.\ \ldots\ .\pi_n$ where $\pi_{1\boxed{i}}, \pi_{2\boxed{j}}, \ldots, \pi_{n\boxed{k}}$ or $\pi_{1\boxed{i}}, \pi_{2\boxed{j}}, \ldots, \pi_{n-1\boxed{k}}$ are unique identifier segments

c. The *result of removing the prefix* π_1 *from a path* $\pi_1.\pi_2$, in symbols $\pi_1.\pi_2\backslash\pi_1$, is π_2

We can now characterize $[T]_\eta$ as in (46).

(46) The *result of relabelling a type* T *with a relabelling,* η*, for* T, $[T]_\eta$, is defined by carrying out the following on the unique identifier notation of T.

1. if $\pi \in \mathrm{dom}(\eta)$ and π (that is, without unique identifiers) is represented as a path in T, then replace π with $\eta(\pi)$
2. if $\pi_{1\boxed{i}}.\pi_{2\boxed{j}}.\ \ldots\ .\pi_{n\boxed{k}}$ is a unique identifier segmentation in $\mathrm{dom}(\eta)$ then replace it with $\eta(\pi_1)_{\boxed{i}}.(\eta(\pi_1.\pi_2)\backslash\eta(\pi_1))_{\boxed{j}}.\ \ldots\ .(\eta(\pi_1.\pi_2.\ \ldots\ .\pi_n)\backslash\eta(\pi_1.\pi_2.\ \ldots\ .\pi_{n-1}))_{\boxed{k}}$
3. if $\pi_{1\boxed{i}}.\pi_{2\boxed{j}}.\ \ldots\ .\pi_n$ is a unique identifier segmentation in $\mathrm{dom}(\eta)$ then replace it with $\eta(\pi_1)_{\boxed{i}}.(\eta(\pi_1.\pi_2)\backslash\eta(\pi_1))_{\boxed{j}}.\ \ldots\ .\eta(\pi_1.\pi_2.\ \ldots\ .\pi_n)\backslash\eta(\pi_1.\pi_2.\ \ldots\ .\pi_{n-1})$

Note that using this definition in terms of unique identifier notation will automatically account for cases in which the scope of the dependent field has to be adjusted in the relabelling. Consider the record type in (47a) and the relabelling in (47b) which should result in (47c).

(47) a. $\left[\begin{array}{lll} \mathrm{x} & : & \left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{e} & : & \langle \lambda v{:}\textit{Ind}\ .\ \mathrm{dog}(v), \langle \mathrm{x}\rangle\rangle \end{array}\right] \end{array}\right]$

b. x.x $\rightsquigarrow$ y

c. $\left[\begin{array}{lll} \mathrm{y} & : & \textit{Ind} \\ \mathrm{x} & : & \langle \lambda v{:}\textit{Ind}\ .\ \left[\begin{array}{lll} \mathrm{e} & : & \mathrm{dog}(v) \end{array}\right], \langle \mathrm{y}\rangle\rangle \end{array}\right]$

The unique identifier notation for (47a) and (47c) are (48a) and (48b) respectively and (48b) is the result of relabelling (48a) with (47b) according to the definition we have given.

(48) a. $\left[\begin{array}{lll} \mathrm{x} & : & \left[\begin{array}{lll} \mathrm{x}_{\boxed{0}} & : & \textit{Ind} \\ \mathrm{e} & : & \mathrm{dog}(\boxed{0}) \end{array}\right] \end{array}\right]$

b. $\left[\begin{array}{lll} \mathrm{y}_{\boxed{0}} & : & \textit{Ind} \\ \mathrm{x} & : & \left[\begin{array}{lll} \mathrm{e} & : & \mathrm{dog}(\boxed{0}) \end{array}\right] \end{array}\right]$

To understand why this works, see the characterization of unique identifier notation in Chapter 2, p. 79.

Given that we have now found a match, how can we go about updating the shared commitments with the new information represented by the parametric content?

If we are updating (34) with the parametric content (36a) then the result should be (49) where (34) has been embedded under the label 'prev' and the new information provided by the parametric content has been added at the top level of the new type, suitably relabelled so as to pick out the individual named Sam which has been previously introduced.

(49)
$$\left[\begin{array}{ll}
\text{prev:} & \left[\begin{array}{ll}
\text{prev:} & \left[\begin{array}{ll}
\text{prev:} & \left[\begin{array}{ll}
\text{prev:} & \left[\begin{array}{ll}
\multicolumn{2}{l}{\text{prev:}\mathit{Rec}} \\
\text{bg:} & \left[\begin{array}{l} \text{x:}\mathit{Ind} \\ \text{e:named(x, “Dudamel”)} \end{array}\right] \\
\text{fg:} & \left[\text{e:conductor}(\Uparrow\text{bg.x})\right]
\end{array}\right] \\
\text{bg:} & \left[\begin{array}{l} \text{x:}\mathit{Ind} \\ \text{e:named(x, “Beethoven”)} \end{array}\right] \\
\text{fg:} & \left[\text{e:composer}(\Uparrow\text{bg.x})\right]
\end{array}\right] \\
\text{bg:} & \left[\begin{array}{l} \text{x:}\mathit{Ind} \\ \text{e:named(x, “Uchida”)} \end{array}\right] \\
\text{fg:} & \left[\text{e:pianist}(\Uparrow\text{bg.x})\right]
\end{array}\right] \\
\text{bg:} & \left[\begin{array}{l} \text{x:}\mathit{Ind} \\ \text{e:named(x, “Sam”)} \end{array}\right] \\
\text{fg:} & \left[\text{e:singer}(\Uparrow\text{bg.x})\right]
\end{array}\right] \\
\text{bg:} & \left[\begin{array}{ll}
\text{f:} & \left[\begin{array}{l} \text{x=}\Uparrow^2\text{prev.bg.x:}\mathit{Ind} \\ \text{e=}\Uparrow^2\text{prev.bg.e:named(x, “Sam”)} \end{array}\right] \\
\multicolumn{2}{l}{\text{a=}\Uparrow\text{prev}^5\text{:}\mathit{Rec}}
\end{array}\right] \\
\text{fg:} & \left[\text{e:leave}(\Uparrow\text{bg.f.x})\right]
\end{array}\right]$$

Note that this both achieves a link to a previous mention of Sam and simultaneously ensures that Sam is the most salient individual in shared commitments in virtue of the new mention.

We can achieve this update by the following method. Suppose that T_{comm} is the type representing shared commitments that we wish to update with a parametric content given in (50).

(50)
$$\left[\begin{array}{lcl} \text{bg} & = & T_{\mathrm{bg}} \\ \text{fg} & = & f \end{array}\right]$$

where $f : (T_{\mathrm{bg}} \rightarrow \mathit{RecType})$. We need to find a relabelling, η, of T_{bg} such that $T_{\mathrm{comm}} \sqsubseteq [T_{\mathrm{bg}}]_\eta$. Suppose that we have a record, r_{comm}, of type T_{comm}. We can use this together with η to anchor T_{bg}. The result of the anchoring is notated as $T_{\mathrm{bg}} \parallel_\eta r_{\mathrm{comm}}$. The operation $T \parallel_\eta r$ replaces fields in T, $\left[\ell{:}T'\right]$, such that ℓ is in the domain of the relabelling, η, and for which η returns a path, π, in r such that $r.\pi : T'$ with a manifest field $\left[\ell{=}r.\pi{:}T'\right]$. For example, suppose that T_{bg} is (51a), $r_{\mathrm{comm}} : T_{\mathrm{comm}}$ and η is (51b). Then $T_{\mathrm{bg}} \parallel_\eta r_{\mathrm{comm}}$ is (51c).

(51) a. $\left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Sam”)} \end{array}\right]$

b. x $\rightsquigarrow$ bg.x
e $\rightsquigarrow$ bg.e

c. $\left[\begin{array}{lll} \text{x=}r_{\text{comm}}\text{.bg.x} & : & \textit{Ind} \\ \text{e=}r_{\text{comm}}\text{.bg.e} & : & \text{named(x, “Sam”)} \end{array}\right]$

In the type of the updated shared commitments the background will be the background of the parametric content anchored to the previous shared commitments and the foreground will be the result of applying the function which is the foreground of the parametric content to this background. The updated type of the shared commitments will thus be that given in (52).

(52) $\left[\begin{array}{lll} \text{prev} & : & T_{\text{comm}} \\ \text{bg} & : & T_{\text{bg}} \parallel_\eta \text{prev} \\ \text{fg} & : & f(\text{bg}) \end{array}\right]$

Recall that a dependent field in a record type normally represented as $[\ell = a : T]$ is in actual fact $[\ell : T_a]$ where T_a is a singleton type. The intuition in defining $T \parallel r$ is that for any path, π, in T leading to a type $T.\pi$, $T.\pi$ is replaced by the singleton type constructed from $T.\pi$ and $r.\pi$. This is made precise in (53).

(53) If T is a record type and r is a record, then $T \parallel r$, the *specification (or anchoring) of* T *by* r is a type, T', like T except that if π is a path in both T and r,

1. if $T.\pi$ is a type, then $T'.\pi$ is $(T.\pi)_{r.\pi}$ (that is, if ℓ is the last label in π then $[\ell : T.\pi]$ is replaced by $[\ell = r.\pi : T.\pi]$ at the end of π in T)
2. if $T.\pi$ is $\langle f, \langle \pi_1, \ldots, \pi_n \rangle \rangle$, then $T'.\pi$ is $\langle f', \langle \pi_1, \ldots, \pi_n \rangle \rangle$ where for any $a_1, \ldots, a_n$, $f'(a_1) \ldots (a_n)$ is defined iff $f(a_1) \ldots (a_n)$ is defined and $f'(a_1) \ldots (a_n) = (f(a_1) \ldots (a_n))_{r.\pi}$

Note that according to this definition if the type at the end of path π is already a singleton type, that is, the field is already manifest, this will add an additional identity constraint. Recall that $c : (T_a)_b$ iff $c : T_a$ and $c = b$. Thus $c : (T_a)_b$ requires $a = b = c$.

A variant of this notion of specification is default specification which will only require specification of fields which are not already specified, that is, manifest fields will remain unchanged. This variant is given in (54).

(54) If T is a record type and r is a record, then $T//r$, the *default specification (or anchoring) of* T *by* r is a type, T', like T except that if π is a path in both T and r,

1. if $T.\pi$ is a type but not a singleton type, then $T'.\pi$ is $(T.\pi)_{r.\pi}$
2. if $T.\pi$ is $\langle f, \langle\pi_1, \ldots, \pi_n\rangle\rangle$, then $T'.\pi$ is $\langle f', \langle\pi_1, \ldots, \pi_n\rangle\rangle$ where for any $a_1, \ldots, a_n$, $f'(a_1)\ldots(a_n)$ is defined iff $f(a_1)\ldots(a_n)$ is defined and $f'(a_1)\ldots(a_n) = (f(a_1)\ldots(a_n))_{r.\pi}$ if $f(a_1)\ldots(a_n)$ is not a singleton type and otherwise $f'(a_1)\ldots(a_n) = f(a_1)\ldots(a_n)$

Types can also be specified by records which have different paths to the type by using a relabelling. Thus $T \parallel_\eta r$ anchors the value of a path, π, in T with the value of $\eta(\pi)$ in r. This is defined in (55).

(55) If T is a record type, r is a record and η is a relabelling of T whose range is included in $\mathrm{paths}(r)$, then $T \parallel_\eta r$, the *specification (or anchoring) of* T *by* r *relative to* η is a type, T', like T except that if π is in the domain of η,

1. if $T.\pi$ is a type, then $T'.\pi$ is $(T.\pi)_{r.\eta(\pi)}$
2. if $T.\pi$ is $\langle f, \langle\pi_1, \ldots, \pi_n\rangle\rangle$, then $T'.\pi$ is $\langle f', \langle\pi_1, \ldots, \pi_n\rangle\rangle$ where for any $a_1, \ldots, a_n$, $f'(a_1)\ldots(a_n)$ is defined iff $f(a_1)\ldots(a_n)$ is defined and $f'(a_1)\ldots(a_n) = (f(a_1)\ldots(a_n))_{r.\eta(\pi)}$

In (56) we provide two simple examples.

(56) a. let η_1 be characterized by

$\ell_1 \rightsquigarrow \ell_3$
$\ell_2 \rightsquigarrow \ell_4$

b. then $\begin{bmatrix} \ell_1 & : & T_1 \\ \ell_2 & : & T_2 \end{bmatrix} \parallel_{\eta_1} \begin{bmatrix} \ell_3 & = & a \\ \ell_4 & = & b \end{bmatrix} = \begin{bmatrix} \ell_1{=}a & : & T_1 \\ \ell_2{=}b & : & T_2 \end{bmatrix}$

c. let η_2 be characterized by

$\ell_5.\ell_1 \rightsquigarrow \ell_7.\ell_3$
$\ell_5.\ell_2 \rightsquigarrow \ell_8.\ell_4$

d. then $\begin{bmatrix} \ell_5 & : & \begin{bmatrix} \ell_1 & : & T_1 \\ \ell_2 & : & T_2 \end{bmatrix} \\ \ell_6 & : & T_3 \end{bmatrix} \parallel_{\eta_2} \begin{bmatrix} \ell_7 & = & \begin{bmatrix} \ell_3 & = & a \end{bmatrix} \\ \ell_8 & = & \begin{bmatrix} \ell_4 & = & b \\ \ell_9 & = & c \end{bmatrix} \end{bmatrix} =$

$\begin{bmatrix} \ell_5 & : & \begin{bmatrix} \ell_1{=}a & : & T_1 \\ \ell_2{=}b & : & T_2 \end{bmatrix} \\ \ell_6 & : & T_3 \end{bmatrix}$

We can tie all this together in a single action rule, given in (57), a preliminary version which we will revise slightly later in the light of other accommodation functions which we will introduce in Section 4.4.

(57) AccGB – preliminary version

$$\frac{s_{i,A} :_A T_{\text{inf}} \;\; T_{\text{inf}} \sqsubseteq \left[\text{shared:}\left[\text{commitments:}\textit{RecType}\right]\right] \;\; u^*.\text{cont} :_A T_{\text{utt}} \;\; T_{\text{utt}} \sqsubseteq \begin{bmatrix}\text{bg:}\textit{RecType}\\ \text{fg:(bg}\rightarrow\textit{RecType})\end{bmatrix}}{s_{i+1,A} :_A T_{\text{inf}}\boxed{\wedge}\left[\text{shared:}\left[\text{commitments=}\begin{bmatrix}\text{prev:}s_{i,A}.\text{shared.commitments}\\ \text{bg:}u^*.\text{cont.bg}\|_\eta\text{prev}\\ \text{fg:}u^*.\text{cont.fg(bg)}\end{bmatrix}\text{:}\textit{RecType}\right]\right]} \; \eta$$

where η is a relabelling for u^*.cont.bg with range included in $s_{i,A}$.shared.commitments

This action rule requires that an agent, A, judges her current information state, $s_{i,A}$, to be of type, T, which includes shared commitments. Furthermore, A judges the content of the current utterance, u^*, to be a parametric content which will yield a record type when applied to a context of the appropriate background type. Given an appropriate relabelling, η, of the background type of the parametric content which associates it with elements of the shared commitments in the current information state, A is licensed (afforded) to make the judgement that her next information state is like the current one except that the shared commitments have been changed so that the current shared commitments are marked as previous, the background of the parametric content anchored to those previous shared commitments constrains a background situation under the label 'bg' and a foreground situation is added, under the label 'fg', resulting from applying the (foreground of) the parametric content to the background thus introduced. (57) abstracts away from the problem of exactly how to choose an appropriate relabelling, η.

4.4 Proper names, salience and accommodation

What we have presented so far enables us to find a match for presuppositions introduced by a parametric content when such a match is present in shared commitments. Suppose there is more than one such match. In that case there will be a choice of relabellings η. In this case we may wish to choose the relabelling that corresponds to a match with the most salient match in terms of recency of introduction into the shared commitments. Technically, this means that we choose the relabelling which introduces paths with the least number of occurrences of 'prev'. Note that the most recent match may be anchored to a match that was introduced earlier in the manner we have just described. There may be other factors than recency which contribute to salience, for example, the kinds of factors that are discussed in centering theory (Joshi and Weinstein, 1981; Grosz *et al.*, 1983, 1995; Walker *et al.*, 1998; Poesio *et al.*, 2004). We will leave it to future work to give a more detailed account of saliency in the current framework.

What happens when there is no match for *Sam* in the shared commitments? Here we need some kind of accommodation in order to use the parametric content to update the gameboard. There are two kinds of accommodation we will consider. The first is where the agent knows of a person named Sam independently of the current conversation. That is, a match for *Sam* can be found in the agent's resources corresponding to long term memory. We will not attempt a detailed account of the stucture of long term memory. We assume that it is complex and constantly in flux not only in terms of new information being added but also in terms of what is salient in the old information, depending on which part of the memory is being focussed on at any particular time. Here we will content ourselves with a simple model of long term memory as a record type of a similar kind to that we have proposed for shared commitments. This means that the techniques we need for matching will be the same as those discussed above. In reality the notion of salience with respect to long term memory will be a good deal more complicated than salience with respect to the shared commitments on the dialogue gameboard. You have to take into account not only recency but also likelihood based on other knowledge that it is this particular Sam that is being referred to. For example, if you believe that your interlocutor could not possibly know of the Sam in your memory who is otherwise the most likely candidate you should not choose that Sam as a match. Choosing an appropriate match involves a great deal of world knowledge and common sense. We will ignore these matters and concentrate our attention on what needs to be done if we find a suitable match. The idea is that if you have failed to find a match in shared commitments on the gameboard but you do find a match in long term memory, then you need to load the item from long term memory into the shared commitments on your gameboard. This is what will constitute accommodation in this case.

We will introduce the notion of a *total information state* (*cf.* Larsson, 2002) which includes a record type corresponding to long term memory, represented by the 'ltm'-field in (58) and a dialogue gameboard, represented by the 'gb'-field in (58). Up until now we have thought of the gameboard as a record type. Now, however, we want to be able to make links from the gameboard to long term memory and we will achieve this by making the gameboard be a dependent type which maps records (situations) of the type representing long term memory to the record type representing the gameboard. Thus a total information state will be of the type (58).

(58) $\begin{bmatrix} \text{ltm} & : & \textit{RecType} \\ \text{gb} & : & (\text{ltm}\rightarrow\textit{GameBoard}) \end{bmatrix}$

Here we use *GameBoard* as the type of types which are a subtype of *InfoState* (for example, as defined most recently in Chapter 3, p.147), that is, a gameboard is a type of information states. Formally, this is expressed as in (59).

(59) T : *GameBoard* iff $T \sqsubseteq$ *InfoState*

An example of a type corresponding to long term memory is given in (60).

(60)
$$\begin{bmatrix} \text{id}_0\text{:}\textit{Rec} \\ \text{id}_1\text{:}\begin{bmatrix} \text{x:}\textit{Ind} \\ \text{e:named(x, “Dudamel”)} \end{bmatrix} \\ \text{id}_2\text{:}\begin{bmatrix} \text{e:conductor(}\Uparrow\text{id}_1\text{.x)} \end{bmatrix} \\ \text{id}_3\text{:}\begin{bmatrix} \text{x:}\textit{Ind} \\ \text{e:named(x, “Beethoven”)} \end{bmatrix} \\ \text{id}_4\text{:}\begin{bmatrix} \text{e:composer(}\Uparrow\text{id}_3\text{.x)} \end{bmatrix} \\ \text{id}_5\text{:}\begin{bmatrix} \text{x:}\textit{Ind} \\ \text{e:named(x, “Uchida”)} \end{bmatrix} \\ \text{id}_6\text{:}\begin{bmatrix} \text{x:pianist(}\Uparrow\text{id}_5\text{.x)} \end{bmatrix} \\ \text{id}_7\text{:}\begin{bmatrix} \text{x:}\textit{Ind} \\ \text{e:named(x, “Sam”)} \end{bmatrix} \\ \text{id}_8\text{:}\begin{bmatrix} \text{e:singer(}\Uparrow\text{id}_7\text{.x)} \end{bmatrix} \end{bmatrix}$$

(60) is one way of putting the information in shared commitments represented by (34) into a type corresponding to long term memory. We are assuming that in long term memory information is indexed by unique identifiers modelled here by the labels ‘id_n’ (of which we assume there is a countably infinite stock, one for each natural number, n). It is important that in long term memory paths are persistent under updating, that is, the old paths do not change when we add information to long term memory. This is in contrast to the kind of updating we proposed for the gameboard, adding the label ‘prev’ to the path for the old gameboard. This meant that all paths within the old gameboard were adjusted by an update. When we link from the gameboard to long term memory we want to make sure that the link uses a persistent path which will still be correct if the long term memory should get updated. When long term memory is updated we prefix the path to the new information with the identifier ‘id_{i+1}’, where i is the highest index on an ‘id’-label in the long term memory type we are updating. (This is the same technique we used for ‘e’-labels in our treatment of chart parsing in Chapter 3.) The way of achieving the link is illustrated schematically in (61) where we use M to represent the long term memory (60) and leave out all irrelevant details of the gameboard.

(61)
$$\begin{bmatrix} \text{ltm=}M\text{:}\textit{RecType} \\ \text{gb=}\lambda r\text{:ltm . }\begin{bmatrix} \ldots \\ \text{shared:}\begin{bmatrix} \text{commitments=}\begin{bmatrix} \ldots \\ \text{bg:}\begin{bmatrix} \text{x=}r\text{.id}_7\text{.x:}\textit{Ind} \\ \text{e=}r\text{.id}_7\text{.e:named(x,“Sam”)} \end{bmatrix} \\ \text{fg:}\begin{bmatrix} \text{e:leave(}\Uparrow\text{bg.x)} \end{bmatrix} \end{bmatrix}\text{:}\textit{RecType} \end{bmatrix} \\ \ldots \end{bmatrix}\text{:(ltm}\rightarrow\textit{RecType}\text{)} \end{bmatrix}$$

The intuition expressed in (61) is as follows: given a situation, r, of the type represented by our long term memory, that is one in which a particular appropriate individual is labelled by ‘id_7’, the gameboard will be a type of information state where the background of the parametric content used to update the shared commitments is anchored to 'id_7'. Two agents are aligned in their shared commitments to the extent that we can find an equivalence between the two types which represent their respective view of the shared commitments obtained by applying their respective functions labelled 'gb' to a situation of their respective memory types.

The link represented by the dependence on the long term memory type corresponds to what Kamp (1990); Kamp *et al.* (2011) call an internal anchor. We are representing here how individual roles in an agent's view of shared commitments can be anchored in that agent's long term memory. In a more complete treatment we could in addition make the gameboard depend on a type for the current visual scene and also types for other sensory input. Our use of dependent types and Kamp *et al.*'s use of internal anchors allow us to link different components of cognitive structure. Cognitive structure can also be linked to objects in the external world, giving rise to what Kamp *et al.* call external anchors. Our manifest fields can be used to correspond to their external anchors. Suppose, for example, that we have an individual 'sam' who is named Sam. We can use a manifest field to restrict the long term memory type (60) so that any record ("situation") of that type has 'sam' in the 'id_7.x'-field. This is represented in (62) where for convenience we have omitted all but the 'id_7'-field in (60).

(62) $$\begin{bmatrix} \ldots \\ \text{id}_7 : \begin{bmatrix} \text{x=sam}:\textit{Ind} \\ \text{e:named(x, “Sam”)} \end{bmatrix} \\ \ldots \end{bmatrix}$$

If M in (61) is the type (62) then for any $r : M$, it will be the case that $r.\text{id}_7.\text{x}$ will be 'sam'. Thus the shared commitment is that 'sam' leaves. Given that manifest fields can occur in any record type, this kind of external anchoring is not restricted to long term memory but could also be directly in the gameboard if that is desired.

Let us now consider how the update of a gameboard dependent on long term memory can be carried out when there is a match between the parametric content used for updating and an item in long term memory. Suppose that agent A's current total information state, $s^{\text{tot}}_{i,A}$, is of the type in (63).

(63) $$\begin{bmatrix} \text{ltm} & : & \textit{RecType} \\ \text{gb}=\lambda r{:}\text{ltm} \,.\, T_{\text{gb}}(\!(r)\!) & : & (\text{ltm}\rightarrow\textit{RecType}) \end{bmatrix}$$

and that we wish to update this with the parametric content, f, given in (64) (where $T_{\text{bg}} \sqsubseteq \begin{bmatrix}\text{x}:\textit{Ind}\end{bmatrix}$).

(64) $$\begin{bmatrix} \text{bg} & = & T_{\text{bg}} \\ \text{fg} & = & \lambda r{:}T_{\text{bg}} \,.\, T_{\text{upd}}(\!(r)\!) \end{bmatrix}$$

In order to find a match between f.bg, that is, T_{bg} and $s^{\text{tot}}_{i,A}$.ltm (that is, to ascertain that the presupposition associated with the parametric content is met by the long term memory of the current total information state) we need to find a relabelling, η, of T_{bg} such that (65) holds.

(65) $s^{\text{tot}}_{i,A}.\text{ltm} \sqsubseteq [T_{\text{bg}}]_\eta$

Then we can derive (66) as a type of the updated total information state.

(66)
$$\left[\begin{array}{l} \text{ltm}=s^{\text{tot}}_{i,A}.\text{ltm}:\mathit{RecType} \\ \text{gb}=\lambda r{:}\text{ltm}\ .\ (T_{\text{gb}}(\!(r)\!)\boxed{\underset{.}{\wedge}} \\ \qquad \left[\text{shared}: \left[\text{commitments}= \left[\begin{array}{l}\text{prev}:s^{\text{tot}}_{i,A}.\text{gb.shared.commitments}\\ \text{bg}:T_{\text{bg}} \parallel_\eta r \\ \text{fg}:f(\text{bg})\end{array}\right]:\mathit{RecType}\right]\right]) \\ \qquad\qquad :(\text{ltm}\rightarrow\mathit{RecType}) \end{array}\right]$$

Thus, for example, if T_{bg} is (33), repeated as (67a), r is a record representing long term memory of type (60), repeated as (67b), and η is the relabelling in (67c), then $T_{\text{bg}} \parallel_\eta r$ is (67d).

(67) a.
$$\left[\mathfrak{c}: \left[\begin{array}{l} \text{f}: \left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Sam”)}\end{array}\right] \\ \text{a}:\mathit{Rec}\end{array}\right]\right]$$

b.
$$\left[\begin{array}{l} \text{id}_0:\mathit{Rec} \\ \text{id}_1:\left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Dudamel”)}\end{array}\right] \\ \text{id}_2:\left[\text{e}:\text{conductor}(\text{id}_1.\text{x})\right] \\ \text{id}_3:\left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Beethoven”)}\end{array}\right] \\ \text{id}_4:\left[\text{e}:\text{composer}(\text{id}_3.\text{x})\right] \\ \text{id}_5:\left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Uchida”)}\end{array}\right] \\ \text{id}_6:\left[\text{x}:\text{pianist}(\text{id}_5.\text{x})\right] \\ \text{id}_7:\left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Sam”)}\end{array}\right] \\ \text{id}_8:\left[\text{e}:\text{singer}(\text{id}_7.\text{x})\right] \end{array}\right]$$

c. $\mathfrak{c}.\text{f.x} \rightsquigarrow \text{id}_7.\text{x}$
$\mathfrak{c}.\text{f.e} \rightsquigarrow \text{id}_7.\text{e}$
$\mathfrak{c}.\text{a} \rightsquigarrow \text{id}_0$

$$\text{d.}\quad \left[\text{c:}\left[\begin{array}{l}\text{f:}\left[\begin{array}{l}\text{x}=r.\text{id}_7.\text{x}:\mathit{Ind}\\ \text{e}=r.\text{id}_7.\text{e:named(x, “Sam”)}\end{array}\right]\\ \text{a}=r.\text{id}_0:\mathit{Rec}\end{array}\right]\right]$$

We can now put all this together as the action rule in (68) on p. 190, which we call AccLTM (“accommodate match with long term memory”).

We have used accommodation from long term memory to represent the kind of accommodation where the agent has a resource which provides a match. In a more complete treatment we could use this technique for accommodation from other available resources such as the visual scene.

We now turn our attention to accommodation where there is no appropriate match with other resources. This corresponds to the case where the hearer does not know any appropriate person named Sam but merely adds that there is a person named Sam to the shared dialogue commitments. The first step in this update is to create a type from the parametric content under consideration so that we can merge it with $\left[\text{prev}:T\right]$, where T is the type representing the current shared commitments. Suppose we are considering the parametric content, ξ, given in (69a). (Recall the corner quotes notation introduced on p. 170.) Then the type we will create from ξ is defined as in (69b) which is identical with (69c).

$$\text{(69)}\quad \text{a.}\quad \xi = \ulcorner\lambda r{:}\left[\begin{array}{l}\text{f:}\left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{e:named(x, “Sam”)}\end{array}\right]\\ \text{a}:\mathit{Rec}\end{array}\right] .\ \left[\text{e:leave}(r.\text{f.x})\right]\urcorner$$

$$\text{b.}\quad \left[\begin{array}{lll}\text{bg} & : & \xi.\text{bg}\\ \text{fg} & : & \left[\ \text{e} \ :\ \xi.\text{fg(bg)}\ \right]\end{array}\right]$$

$$\text{c.}\quad \left[\begin{array}{l}\text{bg:}\left[\begin{array}{l}\text{f:}\left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{e:named(x, “Sam”)}\end{array}\right]\\ \text{a}:\mathit{Rec}\end{array}\right]\\ \text{fg:}\left[\text{e:leave(bg.f.x)}\right]\end{array}\right]$$

Suppose now that the current shared commitments are given by the type in (70).

(68)

$$\frac{s^{\text{tot}}_{i,A} : T_{\text{tot}} \quad T_{\text{tot}} \sqsubseteq \begin{bmatrix}\text{ltm}:\textit{RecType}\\ \text{gb}:\text{ltm}\rightarrow\textit{GameBoard}\end{bmatrix} \quad s^{\text{tot}}_{i,A}.\text{gb}(s^{\text{tot}}_{i,A}.\text{ltm}) \sqsubseteq \begin{bmatrix}\text{shared}:\begin{bmatrix}\text{commitments}:\textit{RecType}\end{bmatrix}\end{bmatrix} \quad u^{*}.\text{cont} :_{A} T_{\text{utt}} \quad T_{\text{utt}} \sqsubseteq \begin{bmatrix}\text{bg}:\textit{RecType}\\ \text{fg}:(\text{bg}\rightarrow\textit{RecType})\end{bmatrix}}{s^{\text{tot}}_{i+1,A} :_{A} \begin{bmatrix}\text{ltm}{=}s^{\text{tot}}_{i,A}.\text{ltm}:\textit{RecType}\\ \text{gb}{=}\lambda r{:}\text{ltm}\ .\ s^{\text{tot}}_{i,A}.\text{gb}(r)\boxed{\dot\wedge}\begin{bmatrix}\text{shared}:\begin{bmatrix}\text{commitments}{=}\begin{bmatrix}\text{prev}:(s^{\text{tot}}_{i,A}.\text{gb}(r)).\text{shared.commitments}\\ \text{bg}:u^{*}.\text{cont.bg}\|_{\eta}\ r\\ \text{fg}:u^{*}.\text{cont.fg}(\text{bg})\end{bmatrix}:\textit{RecType}\end{bmatrix}\end{bmatrix}:(\text{ltm}\rightarrow\textit{GameBoard})\end{bmatrix}}\ \eta$$

where η is a relabelling for u^{*}.cont.bg with range included in $s^{\text{tot}}_{i,A}$.ltm

(70)
$$\begin{bmatrix}\text{prev:} & \begin{bmatrix}\text{prev:} & \begin{bmatrix}\text{prev:}\mathit{Rec} & \\ \text{bg:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Dudamel”)}\end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:conductor(}\Uparrow\text{bg.x)}\end{bmatrix}\end{bmatrix}\\ \text{bg:} & \begin{bmatrix}\text{x:}\mathit{Ind} & \\ \text{fg:} & \begin{bmatrix}\text{e:named(x, “Beethoven”)}\end{bmatrix}\end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:composer(}\Uparrow\text{bg.x)}\end{bmatrix}\end{bmatrix}\\ \text{bg:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Uchida”)}\end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:pianist(}\Uparrow\text{bg.x)}\end{bmatrix}\end{bmatrix}$$

Then the new shared commitments will be (71a) which is (71b).

(71) a.
$$\begin{bmatrix}\text{prev:} & \begin{bmatrix}\text{prev:} & \begin{bmatrix}\text{prev:} & \begin{bmatrix}\text{prev:}\mathit{Rec} & \\ \text{bg:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Dudamel”)}\end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:conductor(}\Uparrow\text{bg.x)}\end{bmatrix}\end{bmatrix}\\ \text{bg:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Beethoven”)}\end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:composer(}\Uparrow\text{bg.x)}\end{bmatrix}\end{bmatrix}\\ \text{bg:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Uchida”)}\end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:pianist(}\Uparrow\text{bg.x)}\end{bmatrix}\end{bmatrix}\end{bmatrix} \underset{\cdot}{\wedge}$$
$$\begin{bmatrix}\text{bg:} & \begin{bmatrix}\text{f:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\\ \text{a:}\mathit{Rec} & \end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:leave(}\Uparrow\text{bg.f.x)}\end{bmatrix}\end{bmatrix}$$

b.
$$\begin{bmatrix}\text{prev:} & \begin{bmatrix}\text{prev:} & \begin{bmatrix}\text{prev:} & \begin{bmatrix}\text{prev:}\mathit{Rec} & \\ \text{bg:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Dudamel”)}\end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:conductor(}\Uparrow\text{bg.x)}\end{bmatrix}\end{bmatrix}\\ \text{bg:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Beethoven”)}\end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:composer(}\Uparrow\text{bg.x)}\end{bmatrix}\end{bmatrix}\\ \text{bg:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Uchida”)}\end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:pianist(}\Uparrow\text{bg.x)}\end{bmatrix}\end{bmatrix}\\ \text{bg:} & \begin{bmatrix}\text{f:} & \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x, “Sam”)}\end{bmatrix}\\ \text{a:}\mathit{Rec} & \end{bmatrix}\\ \text{fg:} & \begin{bmatrix}\text{e:leave(}\Uparrow\text{bg.f.x)}\end{bmatrix}\end{bmatrix}$$

We can now put this together as the action rule in (72) on p. 193, which we call ACCNM ("accommodate no match"). This is the same as ACCLTM in (68) except that in the update for shared commitments there is no anchoring to long term memory.

We can now adjust the preliminary version of ACCGB given in (57) which was the action rule for cases where there is a match on the gameboard so that it matches the general format of action rules for total information states. This is given in (73) on p. 194.

The three action rules for accommodation that we have defined do not specify how we choose the relabelling η. They are governed by the regime in (74).

(74) a. if there is a labelling, η, such that $s^{\text{tot}}_{i,A}$.gb($s^{\text{tot}}_{i,A}$).ltm.shared.commitments $\sqsubseteq [u^{*}.\text{cont.bg}]_{\eta}$, then use AccGB with η

b. else if there is a labelling, η, such that $s^{\text{tot}}_{i,A}$.ltm $\sqsubseteq [u^{*}.\text{cont.bg}]_{\eta}$ then use AccLTM with η

c. else use ACCNM

This account of accommodation for proper names where a new item is allowed to be created in memory when attempts at matching have failed is similar to a proposal by de Groote and Lebedeva (2010) to treat accommodation as error handling when a match has failed to be found. Our information states can be thought of as corresponding to their environment which they consider to be not simply a list of individuals but individuals with their properties, thus providing objects similar to those like the record types which can be found in our information states. One difference between the two proposals, apart from the obvious fact that our aim here has been to embed the theory in a more general theory of dialogue, is that de Groote and Lebedeva use a selection function to select the matches thus apparently assuming an algorithm which yields a unique result. We, on the other hand, talk in terms of matches being licensed and thereby allow for the possibility of non-deterministic selection. What we have in common, though, is that in order to account for the way accommodation is carried out we both add an additional layer to a type theory based semantics and talk in procedural terms of actions to be carried out: we with our action rules and de Groote and Lebedeva with their error handling mechanism.

4.5 Paderewski

Kripke (1979) discusses the case of Peter who hears about a pianist called Paderewski. Later, in a different context, he learns of a Polish national leader and Prime Minister called Paderewski. In reality there was a single (remarkable) man called Paderewski who was both a famous concert pianist and a distinguished statesman. But Peter does not realize this and thinks that he has learned about two distinct people, both named Paderewski. Thus, in our terms, Peter's long term memory might be a subtype of (75) for some natural numbers i, j, k and l.

(72)

$$\frac{s^{\text{tot}}_{i,A} : T_{\text{tot}} \quad T_{\text{tot}} \sqsubseteq \begin{bmatrix}\text{ltm:}\mathit{RecType}\\ \text{gb:ltm}\rightarrow\mathit{GameBoard}\end{bmatrix} \quad s^{\text{tot}}_{i,A}.\text{gb}(s^{\text{tot}}_{i,A}.\text{ltm}) \sqsubseteq \big[\text{shared:}\big[\text{commitments:}\mathit{RecType}\big]\big] \quad u^{*}.\text{cont} :_{A} T_{\text{utt}} \quad T_{\text{utt}} \sqsubseteq \begin{bmatrix}\text{bg:}\mathit{RecType}\\ \text{fg:(bg}\rightarrow\mathit{RecType})\end{bmatrix}}{s^{\text{tot}}_{i+1,A} :_{A} \begin{bmatrix}\text{ltm=}s^{\text{tot}}_{i,A}.\text{ltm:}\mathit{RecType}\\ \\ \text{gb=}\lambda r\text{:ltm . } s^{\text{tot}}_{i,A}.\text{gb}(r)\boxed{\underset{\cdot}{\wedge}}\begin{bmatrix}\text{shared:}\begin{bmatrix}\text{commitments=}\begin{bmatrix}\text{prev:}(s^{\text{tot}}_{i,A}.\text{gb}(r))\text{.shared.commitments}\\ \text{bg:}u^{*}\text{.cont.bg}\\ \text{fg:}u^{*}\text{.cont.fg(bg)}\end{bmatrix}\text{:}\mathit{RecType}\end{bmatrix}\end{bmatrix}\text{:(ltm}\rightarrow\mathit{GameBoard})\end{bmatrix}}$$

(73) AccGB – final version

$$\frac{s^{\text{tot}}_{i,A} :_A T_{\text{tot}} \quad T_{\text{tot}} \sqsubseteq \begin{bmatrix} \text{ltm}:\textit{RecType} \\ \text{gb}:\text{ltm}\rightarrow\textit{GameBoard} \end{bmatrix} \quad s^{\text{tot}}_{i,A}.\text{gb}(s^{\text{tot}}_{i,A}.\text{ltm}) \sqsubseteq \big[\text{shared}:\big[\text{commitments}:\textit{RecType}\big]\big] \quad u^{*}.\text{cont} :_A T_{\text{utt}} \quad T_{\text{utt}} \sqsubseteq \begin{bmatrix} \text{bg}:\textit{RecType} \\ \text{fg}:(\text{bg}\rightarrow\textit{RecType}) \end{bmatrix}}{s^{\text{tot}}_{i+1,A} :_A \begin{bmatrix} \text{ltm}{=}s^{\text{tot}}_{i,A}.\text{ltm}:\textit{RecType} \\ \text{gb}{=}\lambda r{:}\text{ltm}\ .\ s^{\text{tot}}_{i,A}.\text{gb}(r)\boxed{\underset{\cdot}{\wedge}}\begin{bmatrix}\text{shared}: \begin{bmatrix}\text{commitments}{=}\begin{bmatrix} \text{prev}:(s^{\text{tot}}_{i,A}.\text{gb}(r)).\text{shared.commitments} \\ \text{bg}:u^{*}.\text{cont.bg}\|_{\eta}\ (s^{\text{tot}}_{i,A}.\text{gb}(r)).\text{shared.commitments} \\ \text{fg}:u^{*}.\text{cont.fg(bg)} \end{bmatrix}:\textit{RecType}\end{bmatrix}\end{bmatrix}:(\text{ltm}\rightarrow\textit{GameBoard}) \end{bmatrix}}\ \eta$$

where η is a relabelling for u^{*}.cont.bg with range included in $(s^{\text{tot}}_{i,A}.\text{gb}(s^{\text{tot}}_{i,A}.\text{ltm}))$.shared.commitments

$$(75)\quad \left[\begin{array}{l}
\mathrm{id}_i : \left[\begin{array}{l} \mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Paderewski”}) \end{array}\right] \\
\mathrm{id}_j : \left[\mathrm{e}{:}\mathrm{pianist}({\Uparrow}\mathrm{id}_i.\mathrm{x})\right] \\
\mathrm{id}_k : \left[\begin{array}{l} \mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Paderewski”}) \end{array}\right] \\
\mathrm{id}_l : \left[\mathrm{e}{:}\mathrm{statesman}({\Uparrow}\mathrm{id}_k.\mathrm{x})\right]
\end{array}\right]$$

(75) technically allows for the two Paderewskis to be the same individual but if there is nothing in Peter's long term memory that requires them to be the same individual we will count that as corresponding to his view of them as distinct. If Peter were in this state and asked whether the pianist Paderewski and the statesman Paderewski were the same person Peter might reply, "Well, I wouldn't have thought so, but I suppose they could be the same person. I don't know." On being told that the two Paderewskis are in fact the same person he might update his long term memory by carrying out the merge in (76a), that is, his long term memory would now be (76b).

$$(76)\quad \text{a.}\quad \left[\begin{array}{l}
\mathrm{id}_i : \left[\begin{array}{l} \mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Paderewski”}) \end{array}\right] \\
\mathrm{id}_j : \left[\mathrm{e}{:}\mathrm{pianist}({\Uparrow}\mathrm{id}_i.\mathrm{x})\right] \\
\mathrm{id}_k : \left[\begin{array}{l} \mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Paderewski”}) \end{array}\right] \\
\mathrm{id}_l : \left[\mathrm{e}{:}\mathrm{statesman}({\Uparrow}\mathrm{id}_k.\mathrm{x})\right]
\end{array}\right] \underset{\cdot}{\wedge} \left[\begin{array}{l}
\mathrm{id}_i : \left[\mathrm{x}{:}\mathit{Ind}\right] \\
\mathrm{id}_k : \left[\mathrm{x}{=}{\Uparrow}\mathrm{id}_i.\mathrm{x}{:}\mathit{Ind}\right]
\end{array}\right]$$

$$\text{b.}\quad \left[\begin{array}{l}
\mathrm{id}_i : \left[\begin{array}{l} \mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Paderewski”}) \end{array}\right] \\
\mathrm{id}_j : \left[\mathrm{e}{:}\mathrm{pianist}({\Uparrow}\mathrm{id}_i.\mathrm{x})\right] \\
\mathrm{id}_k : \left[\begin{array}{l} \mathrm{x}{=}{\Uparrow}\mathrm{id}_i.\mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Paderewski”}) \end{array}\right] \\
\mathrm{id}_l : \left[\mathrm{e}{:}\mathrm{statesman}({\Uparrow}\mathrm{id}_k.\mathrm{x})\right]
\end{array}\right]$$

Eventually, his long term memory may be restructured to the type in (77) which is set equivalent to that in (76), though not multiset equivalent to it since in any record of this type the individual named Paderewski will only occur once, not twice as in (76).

$$(77)\quad \left[\begin{array}{l}
\mathrm{id}_i : \left[\begin{array}{l} \mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Paderewski”}) \end{array}\right] \\
\mathrm{id}_j : \left[\mathrm{e}{:}\mathrm{pianist}({\Uparrow}\mathrm{id}_i.\mathrm{x})\right] \\
\mathrm{id}_l : \left[\mathrm{e}{:}\mathrm{statesman}({\Uparrow}\mathrm{id}_i.\mathrm{x})\right]
\end{array}\right]$$

We might think of the two types (76b) and (77) as representing two subtly different states of mind which Peter could be in. In (76b) he has two concepts of Paderewski, one concept associated with him being a pianist and perhaps other associated properties, such as practicing hard, wearing tails when he is performing, and so on and the other concept where he is a statesman, and perhaps associated with other properties such as being a dynamic national leader, a driver of hard political bargains or whatever. In (77) he has a single concept of Paderewski including all he knows about him. The first state is perhaps a natural one to be in after just learning that the two Paderewskis are in fact the same, before you have fully assimilated the identity. It is harder to discover contradictions between the two concepts here since it will only be the manifest field linking the two concepts which will reveal the contradiction. Suppose, for example, Peter's concept of the statesman Paderewski has him always late for appointments and pressed for time whereas his concept of the pianist Paderewski has him never late for appointments and not pressed for time. There is no contradiction in the state when Peter believes there to be two Paderewskis. Checking for the inconsistency in the two concept state involves reasoning about the identity expressed by the manifest field. One could imagine a simple consistency checker that does not do this – logically inadequate, of course, but human perhaps. The single concept state could, however, involve a direct conflict between type and its negation which, one imagines, even the simplest of consistency checkers would find. Thus if Peter finds himself in such a state he might need to refine the properties that he was ascribing to the two Paderewskis in order to make the unified concept of the single Paderewski consistent, for example, by modifying the properties to be always late for political meetings and pressed for time in his political life but never late to a musical event and not pressed for time in concerts.

Note that the link that we have expressed between the two concepts in (76b) does not involve anything like an external anchor. An alternative offered us by the type theory to represent that the two Paderewskis are identical is (78), where we are using p to represent the individual Paderewski.

(78) $$\begin{bmatrix} \mathrm{id}_i : \begin{bmatrix} \mathrm{x}{=}p{:}\mathit{Ind} \\ \mathrm{e{:}named(x, ``Paderewski")} \end{bmatrix} \\ \mathrm{id}_j : \begin{bmatrix} \mathrm{e{:}pianist(id}_i\mathrm{.x)} \end{bmatrix} \\ \mathrm{id}_k : \begin{bmatrix} \mathrm{x}{=}p{:}\mathit{Ind} \\ \mathrm{e{:}named(x, ``Paderewski")} \end{bmatrix} \\ \mathrm{id}_l : \begin{bmatrix} \mathrm{e{:}statesman(id}_k\mathrm{.x)} \end{bmatrix} \end{bmatrix}$$

Here the link between Peter's two concepts goes through the world since both his Paderewski concepts are linked to the individual p. If an agent's long term memory is a subtype of (78), then Ind_p figures in the long term memory type (recall that the manifest field $\begin{bmatrix}\mathrm{x}{=}p{:}\mathit{Ind}\end{bmatrix}$ is a notation for $\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}_p\end{bmatrix}$, where Ind_p is a type whose only witness is p (see Appendix 6)). We take this to mean that the agent has a direct way of identifying Paderewski but that he has not in this case become conscious of the identity of the object involved in different perceptions of Paderewski.[4] The situation could be that Peter observes Paderewski on the concert platform in tails and then sees him later in the parliament building. His observations are connected to the same individual although without him realizing that he has observed the same Paderewski twice.

[4]One could choose to interpret such types differently in cognitive terms.

Thus the situation is similar to that decribed for *Hesperus* and *Phosphorus* in Frege (1892). In Frege's case the agent was visually aware of the planet Venus on different occasions, conceived of as the Evening Star (Hesperus) and the Morning Star (Phosphorus) without being aware that the same heavenly body was being observed in the morning as in the evening. The difference between Frege's example and that represented by (78) is that in Frege's case two different proper names were associated with the different observations of the same individual whereas here the same proper name is being used for the same individual, though without awareness that the proper name is being associated with the same individual on both occasions.

Ludlow (2014) discusses Kripke's Paderewski and argues that the reason that proper names can be used to refer to different individuals can be due to the fact that our lexicons are dynamic and that we use different microlanguages on different occasions. In this discussion he is building on previous work by Larson and Ludlow (1993) although in that work the emphasis is on interpreted logical forms (pairs of abstract syntactic representations and semantic values such as truth values for sentences) rather than on local microlanguages constructed for use in a particular situation as argued for on the basis of a number of different kinds of examples in Ludlow (2014). In general the idea of local microlanguages being constructed on the fly during the course of dialogues and for the purposes at hand is something for which I have a great deal of sympathy and have argued for in the past (Cooper and Ranta, 2008; Larsson and Cooper, 2009; Cooper, 2010, 2012b). And indeed Ludlow (2014) is right to argue that proper names provide support for this view of language. The argument is straightforward in the case of proper names and does not involve the kinds of subtleties of meaning variation which can lead some people to suspicion of this view in the case of other words. If somebody says to me at a party, "I'd like to introduce you to my friend Sam" and indeed I have never met Sam before, I can, as a competent speaker of English, immediately form an association between the phonological type "Sam" and the individual to whom I have been introduced. It is obviously not part of my competence as a speaker of English to know all of the individuals in the universe named Sam. Our competence lies rather in our ability to make the connection between the phonological type (a name) and an individual as the need arises. The competence involves a *dynamic* process of acquiring a linguistic coupling of a speech event type with another part of the world and not a *static* knowledge of all the available couplings. Once I have added this pairing, modelled in our terms as a sign type, to my resources, I have in a technical sense modified my language.[5] An advantage of sign-based approaches of the kind we are proposing is that you do not have to resort to subscripts in some logical language in order to distinguish between pairings of the same phonological type with different individuals. This is a trap which Larson and Ludlow (1993) fall into when they claim that there are two (or more) names in such cases distinguished by subscripts in logical form. A disadvantage of this analysis is that no two individuals could have the same name in logical form and thus we would have to use something else to analyze sentences like (79).

(79) My wife's sister, one of my graduate students and our neighbour all have the same name:

[5]In my case the resource is quite likely to disappear again shortly afterwards. People vary in their ability to remember names.

Karin

(79) describes a confusing situation which I have to contend with on a daily basis. If the logical form theory with subscripts were correct this sentence would be necessarily false and one might have expected that the natural way to describe this situation would rather have been (80).

(80) My wife's sister, one of my graduate students and our neighbour all have similar names in that they are pronounced "Karin"

(80), according to my intuitions, is not a natural way of describing the situation. This suggests to me that one would need something in addition to, or in place of, a logical form with subscripts to explain how speakers of natural languages individuate names.

One interpretation of Ludlow's proposal is that when a proper name is used to refer to different individuals, different microlanguages are used for the references to the different individuals. Thus when Elisabet says *Karin* and means her sister, she is using a slightly different language than when she says *Karin* and means our neighbour. While I am much in sympathy with the idea of different microlanguages in general it seems to me that such a proposal could not be quite right. Consider dialogues like (81), a kind of dialogue which is not infrequent in our house.

(81) Elisabet: Karin called
Robin: Karin?
Elisabet: My sister

My utterance in (81) is an example of what is called a clarification request in the dialogue literature (Ginzburg and Cooper, 2004; Ginzburg, 2012, and much other literature). According to that literature one of the uses of a clarification request such as *Karin?* is to ask for further identification of the referent of the use of the proper name in the previous dialogue turn. It might initially seem tempting to regard such a request as being in effect a request for (partial) identification of the microlanguage Elisabet is talking. But if we take that route then we have to ask ourselves what language the clarification request itself is in. Assuming that we have three variants of microlanguages available, one where *Karin* refers to Elisabet's sister, one where it refers to our neighbour and one where it refers to my graduate student, then if the request is in any of those languages the answer to the question is selfevident and it is hard to see why I would ask it. And in particular if I was thinking of Karin, my graduate student, I might be justified in saying that Elisabet's answer was wrong. This, of course, is not at all what is going on. It seems that the clarification request is part of a microlanguage in which *Karin* can be used to refer to any of the three and I am interested in finding out which was meant here. This is the kind of option that might be offered by our sign-based approach where a single (micro)language can contain several different signs with the same phonology but with different contents. The exact treatment of this needs, of course, an account of questions and clarification questions in particular which we will not undertake here.

One can understand, however, why the idea of a single referent for a proper name in a single microlanguage might seem attractive. When Kripke (1979) introduces the puzzle about Peter and Paderewski he is careful to point out the circumstances under which Peter came to the conclusion that there were two Paderewskis. Peter first learns the name Paderewski in connection with the famous pianist. Then: "Later, in a different circle, Peter learns of someone called 'Paderewski' who was a Polish nationalist leader and prime minister." Kripke's example would not have been at all as convincing if Peter had learned about Paderewski, the pianist and Paderewski, the statesman from the same person in the same conversation. Ludlow (2014) makes a similar point in criticising Kripke's construction of the apparent contradiction that Peter believes, namely that Paderewski both is and is not a pianist. "The fallacy involves the conjunction of two sentences that have the appearance of contradicting each other... but they do not contradict because they come from different microlanguages." (p. 148). The fact of the matter is that we do tend to use proper names to refer uniquely within the same dialogue, all other things being equal. Suppose we are involved in a conversation about pianists and have been, say, comparing the relative merits of Paderewski and Ashkenazy, and at some point I say (82)

(82) Paderewski was a leading statesman in Poland

You would naturally infer that I was talking about the same Paderewski, unless I explicitly point out that I intend to refer to a different person with the same name. It is, of course, possible to refer to two different people with the same name within the same dialogue and even within the same sentence, even though it may lead to confusion. The assumption is normally, though, that within the space of a dialogue a name will refer to a unique individual unless it is explicitly stated otherwise. One way of being explicit is to say something like (83)

(83) I know another person named Paderewski

If both dialogue participants are aware of the two people with the same name it is possible to use the names together in a construction which normally requires different intended referents as in (84).[6]

(84) Churchland and Churchland think that replacement of symbol manipulation computer-like devices... with connectionist machines hold (*sic*) great promise

(Globus, 1995, p. 21)

[6] I am grateful to Anders Tolland and Stellan Petersson for calling my attention to the fact that the Churchlands are often referred to as "Churchland and Churchland".

Two people named John engaged in conversation with a third person can refer to each other with the name *John* when addressing the third person without risk of confusion as in (85)

(85)	John E:	I remember John as an inspiring professor when I was a student
	John P:	Well, I remember John as an extremely bright student
	Third person:	I didn't realize you'd known each other that long

When addressing a person you can always use their name as a vocative even if the message you wish to convey involves a person with the same name as in (86).

(86)	A:	John, I'd like to introduce you to my good friend John
	B:	Glad to meet you. Another John, eh?

It is conceivable that somebody would want to argue that all of these cases where the same name is used twice to refer to different people are examples of code-switching between microlanguages within the space of a dialogue or sentence. Since code-switching does take place even between different languages like English and Portuguese within single dialogues and sentences it is hard to say that such an analysis is impossible. However, given that a sign-based analysis of proper names does not require these examples to be cases of code-switching perhaps the onus is on the proponent of the code-switching analysis to motivate this more complex analysis.

Puzzles about proper names and reference such as the Paderewski puzzle and Frege's (1892) original puzzle about *Hesperus* and *Phosphorus* are standardly presented as puzzles about belief reports. Indeed the matters we have discussed in this section do give rise to puzzles in belief reports and we will return to this later. However, we would like to claim that the discussion here shows that the basis of these puzzles does not lie in the analysis of belief reports *per se* but in the nature of communication in dialogue and the resulting organization of memory. While these phenomena seem puzzling from a Fregean or Montagovian formal language perspective, from the point of view of a dialogic approach employing a sign-based analysis they seem to be a natural consequence of the way that communication takes place and knowledge gets stored.

4.6 The interpretation of unbound pronouns

So far in this chapter we have talked about reference associated with uses of proper names and seen that mental states play an important role in explaining how reference works. The sign-based approach to proper names that we have taken is essentially different from the traditional approach to proper names as being something like constants in a logic, since it allows proper name phonologies to be paired with different referents in different utterances. In this way our proper names are behaving more like free variables whose referents are determined by a context. So called deictic or unbound uses of pronouns are often related to free variables in logic. The following question now arises: how does our treatment of proper names relate to our treatment of unbound pronouns? In this section we will give a treatment of unbound pronouns which will also lay a basis for the treatment of binding in Chapter 7.

We will consider how to recreate a simple interpretation of pronouns ranging over individuals, first treating them in a similar way to free variables in logic. The central idea is to use records as *pronominal contexts* which correspond to partial assignments to variables in standard logical treatments. Consider first a simple sentence with a deictic pronoun as in (87).

(87) he left

In our initial pass we will ignore matters of gender to make things simpler. The content of (87) is a type which depends on a context (a situation) which provides a value of the pronoun *he*. Thus it will have a parametric content which is a function from a context assigning a value to the pronoun to a type. In order to do this we will make use of the fact that our contexts are structured.

The contexts that we have used so far have a single component (labelled '$\mathfrak{c}$') which are records of general types corresponding to presuppositions (Beaver and Geurts, 2014) or content not *under discussion* (Ginzburg, 1994, 2012) or not *at issue* (Potts, 2005). For example, (88) puts a constraint on the context (expresses a presupposition) that there is an individual named Sam.

(88) $\left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, "Sam")} \end{array}\right]$

In the treatment of unbound pronouns we will use records that are much more restricted in nature and which correspond to partial assignments to variables. Their labels will all be of the form 'x_i' where i is a natural number and the record types that we will use will be restricted to using a restricted number of non-dependent types. For the current account of singular unbound pronouns we will restrict ourselves to the type *Ind*. Thus (89) is an example of the kind of record type we will use.

(89) $\left[\begin{array}{lll} \text{x}_0 & : & \textit{Ind} \\ \text{x}_1 & : & \textit{Ind} \end{array}\right]$

We will introduce a general type, *Assgnmnt*, of assignments of which (89) will be a subtype.

This is done in (90).

(90) *Assgnmnt* is a basic type. r : *Assgnmnt* iff r : *Rec* and $\mathrm{labels}(r) \subset \{\mathrm{x}_0, \mathrm{x}_1, \ldots\}$

If we wish to restrict assignments to be assignments of individuals we can give the more specific witness condition in (91).

(91) r : *Assgnmnt* iff r : *Rec*, $\mathrm{labels}(r) \subset \{\mathrm{x}_0, \mathrm{x}_1, \ldots\}$ and for any i such that $\mathrm{x}_i \in \mathrm{labels}(r)$, $r.\mathrm{x}_i$: *Ind*

We regard these different kinds of context as separate components of the context as they may be treated differently by compositional semantics, for example, in terms of their projection properties which regulates to what extent contexts required by lower constituents are also required by higher constituents, as discussed, for example, by Potts (2005). We will have parametric contents which look like (92).

(92) $$\ulcorner\lambda c{:}\begin{bmatrix}\mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\\ \mathrm{x}_1{:}\mathit{Ind}\end{bmatrix}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Sam”})\end{bmatrix}\\ \ldots\end{bmatrix}\ .\ \varphi(\!(c)\!)\urcorner$$

We use the label '$\mathfrak{s}$' for the component in the context which assigns entities to occurrences of pronouns and the label '$\mathfrak{c}$', as before, for the component which corresponds to presuppositions. We leave open the possibility of adding further components to the context later. We shall call the component of the context labelled '$\mathfrak{c}$' the *propositional context* since it places constraints on what must hold true in the context. It will be important that the labels used do not overlap with those of the assignment. For this reason we introduce a type *PropCntxt*.

This is done in (93).

(93) *PropCntxt* is a basic type. r : *PropCntxt* iff r : *Rec* and $\mathrm{labels}(r) \cap \{\mathrm{x}_0, \mathrm{x}_1, \ldots\} = \emptyset$

We will now revise the characterization of *Cntxt* in (94) to require both an assignment and a propositional context.

(94) *Cntxt* designates

$$\begin{bmatrix} \mathfrak{s} & : & \textit{Assgnmnt} \\ \mathfrak{c} & : & \textit{PropCntxt} \end{bmatrix}$$

The parametric content for *he* could be that given in (95a), for *left* (ignoring tense) that given in (95b) and their combination, using a variant of S-combination which we will discuss as we progress, is represented in (95c).

(95) a. $\ulcorner\lambda c{:}\textit{Cntxt}\mathbin{\dot\wedge}\big[\mathfrak{s}{:}\big[\mathrm{x}_0{:}\textit{Ind}\big]\big]\ .\ \lambda P{:}\textit{Ppty}\ .\ P(\big[\mathrm{x}{=}c.\mathfrak{s}.\mathrm{x}_0\big])\urcorner$

b. $\ulcorner\lambda c{:}\textit{Cntxt}\ .\ \lambda r{:}\big[\mathrm{x}{:}\textit{Ind}\big]\ .\ \big[\ \mathrm{e}\ :\ \mathrm{leave}(r.\mathrm{x})\ \big]\urcorner$

c. $\ulcorner\lambda c{:}\textit{Cntxt}\mathbin{\dot\wedge}\big[\mathfrak{s}{:}\big[\mathrm{x}_0{:}\textit{Ind}\big]\big]\ .\ \big[\ \mathrm{e}\ :\ \mathrm{leave}(c.\mathfrak{s}.\mathrm{x}_0)\ \big]\urcorner$

Using the notation for merging with record types introduced in Chapter 2, p. 92 we can represent (95) as (96).

(96) a. $\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}{:}\big[\mathrm{x}_0{:}\textit{Ind}\big]\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(\big[\mathrm{x}{=}c.\mathfrak{s}.\mathrm{x}_0\big])\urcorner$

b. $\ulcorner\lambda c{:}\textit{Cntxt}\ .\ \lambda r{:}\big[\mathrm{x}{:}\textit{Ind}\big]\ .\ \big[\ \mathrm{e}\ :\ \mathrm{leave}(r.\mathrm{x})\ \big]\urcorner$

c. $\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}{:}\big[\mathrm{x}_0{:}\textit{Ind}\big]\end{bmatrix}\ .\ \big[\ \mathrm{e}\ :\ \mathrm{leave}(c.\mathfrak{s}.\mathrm{x}_0)\ \big]\urcorner$

We will define ‘SemPron’ to be (97).

(97) $\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}{:}\big[\mathrm{x}_0{:}\textit{Ind}\big]\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(\big[\mathrm{x}{=}c.\mathfrak{s}.\mathrm{x}_0\big])\urcorner$

We will define ‘LexPron(T_{phon})’, where T_{phon} is a phonological type, to be (98).

(98) $\mathrm{Lex}(T_{\mathrm{phon}}, \textit{NP}) \mathbin{\dot\wedge} \big[\mathrm{cnt}{=}\mathrm{SemPron}{:}\textit{PQuant}\big]$

This means that every instance of a pronoun when viewed in isolation as an utterance of a lexical item will involve the label ‘x_0’. However, when we combine phrases we will increment the indexes on ‘x’ in one of the constituents so that no two occurrences of pronouns will have the same index. This will be done in our combination rule based on the S-combinator (and would also need to be done in other combination rules if they were added). In order to define this we will use the notation π_i to represent a path in which the last label is indexed with i. If T is a record type we characterize the maximum index associated with a path of the form π (without the index), $\max_\pi(T)$, as (99).

(99) $\max(\{i \mid \pi_i \in \text{paths}(T)\})$

Here for convenience we assume that $\max(\emptyset)$ is -1 and that $\max$ picks out the highest of a non-empty set of natural numbers, as normal. This simplifies our definition of incrementation. We use the notation $[T]_{\pi_i \rightsquigarrow \pi_{f(i)}}$ to represent the result of relabelling each path in T of the form π_i where i is a natural number with $\pi_{f(i)}$ where f is a function from natural numbers to natural numbers. We characterize the incrementation of the π-indexes in a record type T_1 with respect to another record type T_2, $\text{incr}_\pi(T_1, T_2)$, is (100).

(100) $[T_1]_{\pi_i \rightsquigarrow \pi_{i+max_\pi(T_2)+1}}$

Similarly, we characterize the incrementation of the π-indexes of a dependent type, $\lambda v\!:\!T\ .\ \varphi(\!(v)\!)$ with respect to a record type, T', that is, (101a), as (101b).

(101) a. $\text{incr}_\pi(\lambda v\!:\!T\ .\ \varphi(\!(v)\!), T')$

b. $\lambda v\!:\!\text{incr}_\pi(T, T')\ .\ \text{incr}_\pi(\varphi(\!(v)\!), T')$

We can now modify the combination rule in (24) to take account of the incrementation of free pronouns. This is given in (102).

(102) If $\alpha : \begin{bmatrix}\text{bg:}\textit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow(T_1 \rightarrow T_2))\end{bmatrix}$ and $\beta : \begin{bmatrix}\text{bg:}\textit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow T_1)\end{bmatrix}$ then the *combination of* α *and* β *based on functional application*, $\alpha@\beta$, is

$$\ulcorner\lambda c{:}[\alpha.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{f}}\ \dot{\wedge}\,\text{incr}_{\mathfrak{s}.\text{x}}([\beta.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{a}}, \alpha.\text{bg})\ .\\ [\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{f}}(c)(\text{incr}_{\mathfrak{s}.\text{x}}([\beta.\text{fg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{a}}, \alpha.\text{bg})(c))\urcorner$$

While each pronoun consider on its own will have a content involving the label 'x_0' whenever constituents are combined relabelling will take place that ensures that none of the 'x_i' in the contents of the two constituents are the same. Thus considering the boy and the dog, a content for (103a) will be (103b).

(103) a. he hugged it

$$\text{b. } \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\\ \mathrm{x}_1{:}\textit{Ind}\end{bmatrix}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & \mathrm{hug}(c.\mathfrak{s}.\mathrm{x}_0,\ c.\mathfrak{s}.\mathrm{x}_1)\end{bmatrix}\urcorner$$

In order to achieve this we need to introduce resources for transitive verbs as in (104).

(104) a. If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \textit{Ind}, \textit{Ind}\rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}} . \lambda q{:}\textit{Quant} . \lambda r_1{:}\begin{bmatrix}\mathrm{x}{:}\textit{Ind}\end{bmatrix} . q(\lambda r_2{:}\begin{bmatrix}\mathrm{x}{:}\textit{Ind}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & p(r_1.\mathrm{x}, r_2.\mathrm{x})\end{bmatrix})\urcorner$$

b. If T_{phon} is a phonological type, T_{bg} a record type (for context) and p is a predicate with arity $\langle \textit{Ind}, \textit{Ind}\rangle$, then $\text{Lex}_{\text{TransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$ is

$$\text{Lex}(T_{\text{phon}}, V_t) \wedge \begin{bmatrix}\text{cnt=SemTransVerb}(T_{\text{bg}}, p){:}\textit{PPpty}\end{bmatrix}$$

c. $\text{Lex}_{\text{TransVerb}}$(“hug”, *Rec*, hug)

A potential advantage of using record types to characterize pronominal contexts rather than variable assignments is that we can add further information represented by the pronoun such as gender. Thus a simple treatment of gender for (104) might be given by making the content be (105).

$$(105)\ \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\\ \mathrm{x}_1{:}\textit{Ind}\end{bmatrix}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\begin{bmatrix}\textit{PropCntxt}\\ \mathrm{e}{:}\mathrm{male}(\Uparrow^2\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\textit{PropCntxt}\\ \mathrm{e}{:}\mathrm{neuter}(\Uparrow^3\mathfrak{s}.\mathrm{x}_1)\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & \mathrm{hug}(c.\mathfrak{s}.\mathrm{x}_0,\ c.\mathfrak{s}.\mathrm{x}_1)\end{bmatrix}\urcorner$$

There are some complications with this simple idea when it comes to the interpretation of pronouns which are bound by quantifiers. Even for deictic pronouns there are problems determining which predicates should be used in a semantic treatment of gender. Even for a language like English which apparently has semantic gender (as opposed to grammatical gender like German or French), neuter can be used for objects which do not have gender (like tables) and for animals other than humans which do have gender and which can be referred to with masculine and feminine pronouns. We will not explore this further here.

4.7 Summary of resources introduced

Items that are new since Chapter 3 are marked "**New!**" and items that have been revised since Chapter 3 are marked "**Revised!**". We have included some items for completeness which were not explicitly introduced in the text.

4.7.1 Universal grammar resources

4.7.1.1 Types

Loc — a basic type

l : *Loc* iff l is a region in three dimensional space

Phon — a basic type

e : *Phon* iff e is a phonological event

$$\textit{SEvent} \text{ — } \left[\begin{array}{lll} \text{e-loc} & : & \textit{Loc} \\ \text{sp} & : & \textit{Ind} \\ \text{au} & : & \textit{Ind} \\ \text{e} & : & \textit{Phon} \\ \text{c}_{\text{loc}} & : & \text{loc(e,e-loc)} \\ \text{c}_{\text{sp}} & : & \text{speaker(e,sp)} \\ \text{c}_{\text{au}} & : & \text{audience(e,au)} \end{array}\right] \text{ (as in Chapter 2)}$$

Assgnmnt **New!** — a basic type

r : *Assgnmnt* iff r : *Rec* and $\mathrm{labels}(r) \subset \{\mathrm{x}_0, \mathrm{x}_1, \ldots\}$

PropCntxt **New!** — a basic type

r : *PropCntxt* iff r : *Rec* and $\mathrm{labels}(r) \cap \{\mathrm{x}_0, \mathrm{x}_1, \ldots\} = \emptyset$

$$\textit{Cntxt} \textbf{ New! } \text{— } \left[\begin{array}{lll} \mathfrak{s} & : & \textit{Assgnmnt} \\ \mathfrak{c} & : & \textit{PropCntxt} \end{array}\right]$$

CntxtType **New!** — a basic type

T : *CntxtType* iff $T \sqsubseteq$ *Cntxt*

Ppty — ($\left[\text{x}:\textit{Ind}\right]$→*RecType*)

$$\textit{PPpty} \textbf{ New! } \text{— } \left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{Ppty}) \end{array}\right]$$

Quant — (*Ppty*→*RecType*)

$$\textit{PQuant} \textbf{ New! } \text{— } \left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{Quant}) \end{array}\right]$$

QuantDet **New!** — (*Ppty*→*Quant*)

PQuantDet **New!** — $\begin{bmatrix} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{QuantDet}) \end{bmatrix}$

PRecType **New!** — $\begin{bmatrix} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{RecType}) \end{bmatrix}$

Cont **Revised!** — *PRecType*∨*PPpty*∨*PQuant*∨*PQuantDet*

Cat — a basic type

s, np, det, n, v, vp : *Cat*

Syn — $\begin{bmatrix} \text{cat} & : & \textit{Cat} \\ \text{daughters} & : & \textit{Sign}^* \end{bmatrix}$

Sign — a basic type

σ : *Sign* iff σ : $\begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{syn} & : & \textit{Syn} \\ \text{cont} & : & \textit{Cont} \end{bmatrix}$

SignType — a basic type

T : *SignType* iff $T \sqsubseteq$ *Sign* (as in Chapter 2)

S — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=s:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

NP — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=np:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

Det — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=det:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

N — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=n:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

V — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=v:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

VP — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=vp:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

NoDaughters — $\begin{bmatrix}\text{syn:}\begin{bmatrix}\text{daughters=}\varepsilon\text{:}\textit{Sign}^*\end{bmatrix}\end{bmatrix}$

4.7.1.2 Predicates

(as in Chapter 3)

4.7.1.3 Lexicon

Lex

If T_{phon} is a phonological type (that is, $T_{\text{phon}} \sqsubseteq$ *Phon*) and T_{sign} is a sign type (that is, $T_{\text{sign}} \sqsubseteq$ *Sign*), then we shall use Lex(T_{phon}, T_{sign}) to represent

$$((T_{\text{sign}} \mathbin{\dot\wedge} \left[\text{s-event:}\left[\text{e:}T_{\text{phon}}\right]\right]) \mathbin{\dot\wedge} \textit{NoDaughters})$$

SemCommonNoun(p) **Revised!**

If p is a predicate with arity $\langle \textit{Ind}\rangle$, then SemCommonNoun($p$) is

$$\lambda c{:}\textit{Cntxt}\ .\ \lambda r{:}\left[\text{x:}\textit{Ind}\right]\ .\ \left[\ \text{e} \quad : \quad p(r.\text{x})\ \right]$$

Lex$_{\text{CommonNoun}}$(T_{phon}, p)

If T_{phon} is a phonological type and p is a predicate with arity $\langle \textit{Ind}\rangle$, then Lex$_{\text{CommonNoun}}$($T_{\text{phon}}$, p) is

$$\text{Lex}(T_{\text{phon}}, \textit{N}) \mathbin{\dot\wedge} \left[\text{cont=SemCommonNoun}(p){:}\textit{PPpty}\right]$$

SemPropName(T_{phon}) **Revised!**

If T_{phon} is a phonological type, then SemPropName(T_{phon}) is

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix}\text{x:}\textit{Ind} \\ \text{e:named(x, }T_{\text{phon}})\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(c.\mathfrak{c})\urcorner$$

Lex$_{\text{PropName}}$(T_{phon}) **Revised!**

If T_{phon} is a phonological type,

then Lex$_{\text{PropName}}$(T_{phon}) is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot\wedge} \left[\text{cnt=SemPropName}(T_{\text{phon}}){:}\textit{PQuant}\right]$$

SemPron **New!**

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt} \\ \mathfrak{s}{:}\left[\text{x}_0{:}\textit{Ind}\right]\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(\left[\text{x=}c.\mathfrak{s}.\text{x}_0\right])\urcorner$$

LexPron(T_{phon}) **New!**

If T_{phon} is a phonological type, then LexPron(T_{phon}) is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot\wedge} \left[\text{cont=SemPron:}\textit{PQuant}\right]$$

SemIndefArt **Revised!**

$$\ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda Q{:}\mathit{Ppty}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q & : & \mathit{Ppty} \\ \text{scope}{=}P & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{bmatrix}\urcorner$$

$\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{Det}) \wedge \big[\text{cont}{=}\text{SemIndefArt}{:}\mathit{PQuantDet}\big]$$

SemIntransVerb(T_{bg}, p) **New!**

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}\rangle$, then SemIntransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \lambda r{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \big[\ \text{e} \ :\ p(r.\text{x})\ \big]\urcorner$$

$\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$ **New!**

If T_{phon} is a phonological type, T_{bg} a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}\rangle$, then $\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$ is

$$\text{Lex}(T_{\text{phon}}, V_i) \wedge \big[\text{cnt}{=}\text{SemIntransVerb}(T_{\text{bg}}, p){:}\mathit{PPpty}\big]$$

SemTransVerb(T_{bg}, p) **New!**

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}, \mathit{Ind}\rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \lambda \mathcal{Q}{:}\mathit{Quant}\ .\ \lambda r_1{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \mathcal{Q}(\lambda r_2{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \big[\ \text{e}\ :\ p(r_1.\text{x}, r_2.\text{x})\ \big])\urcorner$$

$\text{Lex}_{\text{TransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$ **New!**

If T_{phon} is a phonological type, T_{bg} a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}, \mathit{Ind}\rangle$, then $\text{Lex}_{\text{TransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$ is

$$\text{Lex}(T_{\text{phon}}, V_t) \wedge \big[\text{cnt}{=}\text{SemTransVerb}(T_{\text{bg}}, p){:}\mathit{PPpty}\big]$$

SemBe **Revised!**

$$\lambda c{:}\mathit{Cntxt}\ .\ \lambda \mathcal{Q}{:}\mathit{Quant}\ .\ \lambda r_1{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \mathcal{Q}(\lambda r_2{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \begin{bmatrix} \text{x}{=}r_2.\text{x}, r_1.\text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{be(x)} \end{bmatrix})$$

$\text{Lex}_{\text{be}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{be}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, V) \wedge \big[\text{cont}{=}\text{SemBe}{:}(\mathit{Quant}{\rightarrow}\mathit{Ppty})\big]$$

4.7.1.4 Constituent structure

RuleDaughters($T_{\text{daughters}}$, T_{mother})

If T_{mother} is a sign type and $T_{\text{daughters}}$ is a type of strings of signs then

$$\text{RuleDaughters}(T_{\text{daughters}}, T_{\text{mother}})$$

is

$$\lambda u{:}T_{\text{daughters}}\ .\ T_{\text{mother}} \underset{\cdot}{\wedge} \left[\text{syn:}\left[\text{daughters}{=}u{:}T_{\text{daughters}}\right]\right]$$

ConcatPhon

$$\lambda u{:}\left[\text{s-event:}\left[\text{e:}\mathit{Phon}\right]\right]^{+}\ .$$
$$\left[\ \text{s-event}\ :\ \left[\ \text{e=concat}_i(u[i].\text{s-event.e})\ :\ \mathit{Phon}\ \right]\ \right]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$

If T_{mother} is a sign type and $T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$ are sign types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1} \ldots T_{\text{daughter}_n}$$

represents

$$\text{RuleDaughters}(T_{\text{mother}}, T_{\text{daughter}_1}{}^\frown \ldots {}^\frown T_{\text{daughter}_n}) \underset{\cdot}{\wedge} \text{ConcatPhon}$$

$\alpha @ \beta$ **New!**

If $\alpha : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow(T_1 \rightarrow T_2))\end{bmatrix}$ and $\beta : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow T_1)\end{bmatrix}$ then the *combination of* α *and* β *based on functional application*, $\alpha @ \beta$, is

$$\ulcorner \lambda c{:}[\alpha.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}} \underset{\cdot}{\wedge} \text{incr}_{\mathfrak{s}.\text{x}}([\beta.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}, \alpha.\text{bg})\ .$$
$$[\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}(c)(\text{incr}_{\mathfrak{s}.\text{x}}([\beta.\text{fg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}, \alpha.\text{bg})(c))\urcorner$$

ContForwardApp(T_{arg}, T_{res}) **Revised!**

If T_{arg} and T_{res} are types, then ContForwardApp(T_{arg}, T_{res}) is

$$\lambda u{:}\left[\text{cont:}\begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow(T_{\text{arg}} \rightarrow T_{\text{res}}))\end{bmatrix}\right]^\frown \left[\text{cont:}\begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow T_{\text{arg}})\end{bmatrix}\right]\ .$$
$$\left[\text{cont=}u[0].\text{cont}@u[1].\text{cont:}\begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:}(\text{bg}\rightarrow T_{\text{res}})\end{bmatrix}\right]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \mid T'_{\text{daughter}_1}(T'_{\text{daughter}_2} : T_{\text{arg}}) : T_{\text{res}}$

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and T_{arg} and T_{res} are content types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \mid T'_{\text{daughter}_1}(T'_{\text{daughter}_2} : T_{\text{arg}}) : T_{\text{res}}$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \underset{\cdot}{\wedge} \text{ContForwardApp}(T_{\text{arg}}, T_{\text{res}})$$

4.7.1.5 Action rules

(as in Chapter 3)

4.7.2 Universal speech act resources

(as in Chapter 2)

4.7.3 Universal dialogue resources

4.7.3.1 Types

$$\textit{InfoState} \text{ — } \begin{bmatrix} \text{private} & : & \begin{bmatrix} \text{agenda} & : & \text{list}(\textit{RecType}) \end{bmatrix} \\ \text{shared} & : & \begin{bmatrix} \text{latest-utterance} & : & \textit{Sign}^* \\ \text{commitments} & : & \textit{RecType} \end{bmatrix} \end{bmatrix}$$

$$\textit{InitInfoState} \text{ — } \begin{bmatrix} \text{private} & : & \begin{bmatrix} \text{agenda=[ ]} & : & \text{list}(\textit{RecType}) \end{bmatrix} \\ \text{shared} & : & \begin{bmatrix} \text{latest-utterance=}\varepsilon & : & \textit{Sign}^* \\ \text{commitments=}\textit{Rec} & : & \textit{RecType} \end{bmatrix} \end{bmatrix}$$

GameBoard **New!** — a basic type

T : *GameBoard* iff $T \sqsubseteq$ *InfoState*

$$\textit{TotalInfoState} \textbf{ New! } \text{— } \begin{bmatrix} \text{ltm} & : & \textit{RecType} \\ \text{gb} & : & (\text{ltm}\rightarrow\textit{GameBoard}) \end{bmatrix}$$

4.7.3.2 Update functions and action rules

$$\text{f}_{\text{PlanAckAss}}\ \lambda r{:}\textit{InfoState}\ .\ \lambda u{:}\textit{Assertion}\ .\ \begin{bmatrix} \text{private:} & \begin{bmatrix} \text{agenda:} & \begin{bmatrix} \text{fst:} & \begin{bmatrix} \text{s-event:}\textit{SEvent} \wedge\!\!\!\!. \begin{bmatrix} \text{sp=}u\text{.s-event.au:}\textit{Ind} \\ \text{au=}u\text{.s-event.sp:}\textit{Ind} \end{bmatrix} \\ \text{cont=}u\text{.cont:}\textit{Cont} \\ \text{illoc:acknowledge(s-event, cont)} \end{bmatrix} \\ \text{rst=}r\text{.private.agenda:list}(\textit{RecType}) \end{bmatrix} \end{bmatrix} \\ \text{shared:} & \begin{bmatrix}\text{latest-utterance=}u{:}\textit{Assertion}\end{bmatrix} \end{bmatrix}$$

$$\text{PlanAckAss}\ \frac{s_{i,A} :_A T_{\text{curr}} \quad T_{\text{curr}} \sqsubseteq \text{domtype}(\text{f}_{\text{PlanAckAss}}) \quad u^* :_A T_{\text{utt}} \quad T_{\text{utt}} \sqsubseteq \textit{Assertion}}{s_{i+1,A} :_A T_{\text{curr}} \boxed{\wedge\!\!\!\!.}\ (\text{f}_{\text{PlanAccAss}}(s_{i,A})(u^*) \wedge\!\!\!\!.\ \begin{bmatrix}\text{shared:} \begin{bmatrix}\text{latest-utterance:}T_{\text{utt}}\end{bmatrix}\end{bmatrix})}$$

$$\text{f}_{\text{IntegAck}}\ \lambda r{:}\textit{InfoState}\ .\ \lambda u{:}\textit{Acknowledgement}\ .\ \begin{bmatrix} \text{shared:} \begin{bmatrix} \text{commitments=}\begin{bmatrix}\text{prev:}r\text{.shared.commitments}\end{bmatrix} \wedge\!\!\!\!.\ u\text{.cont:}\textit{RecType} \\ \text{latest-utterance=}u{:}\textit{Acknowledgement} \end{bmatrix} \end{bmatrix}$$

$$\text{INTEGACK}\ \frac{s_{i,a} :_A T_{\text{curr}} \quad T_{\text{curr}} \sqsubseteq \text{domtype}(\text{f}_{\text{IntegAck}}) \quad u^* :_A T_{\text{utt}} \quad T_{\text{utt}} \sqsubseteq \textit{Acknowledgement}}{s_{i+1,A} :_A T_{\text{curr}} \boxed{\dot{\wedge}} (\text{f}_{\text{INTEGACK}}(s_{i,A})(u^*) \mathbin{\dot{\wedge}} \left[\text{shared:}\left[\text{latest-utterance:}T_{\text{utt}}\right]\right])}$$

$$\text{EXECTOPAGENDA}\ \frac{s_{i,A} :_A \textit{InfoState} \mathbin{\dot{\wedge}} \left[\text{private:}\left[\text{agenda:}\begin{bmatrix}\text{fst:}\textit{RecType}\\ \text{rst:list}(\textit{RecType})\end{bmatrix}\right]\right]}{:_A s_{i,A}.\text{private.agenda.fst!}}$$

DOWNDATEAGENDA

$$\frac{s_{i,A} :_A T_{\text{curr}} \quad T_{\text{curr}} \sqsubseteq \left[\text{private:}\left[\text{agenda:}\begin{bmatrix}\text{fst:}\textit{RecType}\\ \text{rst:list}(\textit{RecType})\end{bmatrix}\right]\right] \quad u^* :_A s_{i,A}.\text{private.agenda.fst}}{s_{i+1,A} :_A T_{\text{curr}} \boxed{\dot{\wedge}} \left[\text{private:}\left[\text{agenda=}s_{i,A}.\text{private.agenda.rst:list}(\textit{RecType})\right]\right]}$$

ACCGB **New!** See p. 194.

ACCLTM **New!** See p. 190.

ACCNM **New!** See p. 193.

Control regime for accommodation New!

a. if there is a labelling, η, such that $s^{\text{tot}}_{i,A}.\text{gb}(s^{\text{tot}}_{i,A}).\text{ltm.shared.commitments} \sqsubseteq [u^*.\text{cont.bg}]_\eta$, then use AccGB with η

b. else if there is a labelling, η, such that $s^{\text{tot}}_{i,A}.\text{ltm} \sqsubseteq [u^*.\text{cont.bg}]_\eta$ then use AccLTM with η

c. else use ACCNM

4.7.4 English resources

4.7.4.1 Basic types and predicates

Basic phonological types for words

{“Dudamel”, “is”, “a”, “conductor”, “Beethoven”, “composer”, “Uchida”, “pianist”, “aha”, “ok”, “leaves” **New!**, “hugs” **New!**}

Witnesses for basic types

Ind — dudamel, beethoven, uchida : *Ind* **No longer necessary for interpretation of proper names!**

Predicates

with arity $\langle \textit{Ind} \rangle$ {conductor, composer, pianist, leave **New!**}

with arity $\langle \textit{Ind}, \textit{Ind} \rangle$ {hug **New!**}

4.7.4.2 Grammar

Lexical sign types

$\{\mathrm{Lex}_{\mathrm{PropName}}$(“Dudamel”) **Revised!**,
$\mathrm{Lex}_{\mathrm{PropName}}$(“Beethoven”) **Revised!**,
$\mathrm{Lex}_{\mathrm{Pron}}$(“he”) **New!**,
$\mathrm{Lex}_{\mathrm{IndefArt}}$(“a”),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$(“composer”, composer),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$(“conductor”, conductor),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$(“leave”, *Rec*, leave) **New!**,
$\mathrm{Lex}_{\mathrm{TransVerb}}$(“hug”, *Rec*, hug) **New!**,
$\mathrm{Lex}_{\mathrm{be}}$(“is”),
Lex(“ok”, *S*),
Lex(“aha”, *S*) $\}$

Constituent structure rule components

CnstrIsA

$$\lambda u{:}V \mathbin{\dot{\wedge}} \left[\text{s-event:}\left[\text{e:“is”}\right]\right]^\frown NP \mathbin{\dot{\wedge}} \left[\text{syn:}\begin{bmatrix}\text{daughters:}Det \mathbin{\dot{\wedge}} \left[\text{s-event:}\left[\text{e:“a”}\right]\right] \\ {}^\frown N \mathbin{\dot{\wedge}} \left[\text{cont:}Ppty\right]\end{bmatrix}\right].$$
$$VP \mathbin{\dot{\wedge}} \left[\text{cont=}u[2].\text{syn.daughters}[2].\text{cont:}Ppty\right]$$

Constituent structure rules

$\{S \longrightarrow NP\ VP \mid NP'(VP'{:}Ppty){:}RecType,$
$NP \longrightarrow Det\ N \mid Det'(N'{:}Ppty){:}Quant,$
$VP \longrightarrow V\ NP \mathbin{\dot{\wedge}}$ CnstrIsA,
$VP \longrightarrow V\ NP \mid V'(NP'{:}Quant){:}Ppty\}$

4.8 Summary

In this chapter we have looked at the analysis of proper names and unbound pronouns. We started by showing how Montague’s analysis of proper names could be recast in TTR and we showed that there was an advantage in the sign-based approach that we have adopted in accounting for the fact that different individuals can have the same name. Montague’s original analysis did not say anything about the presupposition-like nature of proper names in that they seem to require interlocutors to be able to identify appropriate referents for the use of a proper name from among a number of potential referents which might be available. We showed how this could be treated by introducing parametric contents for proper names and we showed how accommodation phenomena could be accounted for including a simple-minded analysis of salience analyzed in terms of the information states of agents. We discussed Kripke’s puzzle concerning Paderewski and its possible relation to a theory of microlanguages as discussed by Ludlow. While in general we find the idea of microlanguages appealing we suggested that it plays a role in the analysis of proper names in a rather different way to that suggested by Ludlow.

Finally, we gave an analysis of unbound pronouns. Our treatment of proper names and pronouns shows them to be similar in that they can be related to different individuals depending on context. However, they are associated with different components of the context. Proper names add a restriction on the context that the individual referred to has the appropriate name whereas pronouns are associated with labels indexed by natural numbers. These are the labels of the component of the context which keeps track of pronouns. We can think of this component as corresponding to a partial assignment to variables in logic based treatments of pronouns. Just as in logical treatments it will be this assignment based on subsets of the natural numbers which will be operated when we come to the treatment of pronouns bound by quantifiers in Chapter 7.

Chapter 5

Frames and descriptions

5.1 Montague's treatment of common nouns and individual concepts

The treatment of common nouns in Chapter 3 is encapsulated in $\mathrm{Lex}_{\mathrm{CommonNoun}}$ and carried over into Chapter 4 where it was modified to accommodate parametric contents. The idea is that for a common noun such as *dog* there should be a corresponding predicate 'dog' with arity $\langle$*Ind*$\rangle$ as well as a phonological type "dog". Then an utterance event of the type "dog" will be associated with the content in (1a) whose foreground is of type (1b).

(1) a. $\ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda r{:}\big[\mathrm{x}{:}\mathit{Ind}\big]\ .\ \big[\mathrm{e}{:}\mathrm{dog}(r.\mathrm{x})\big]\urcorner$

b. $(\mathit{Cntxt}{\rightarrow}(\big[\mathrm{x}{:}\mathit{Ind}\big] \rightarrow \mathit{RecType}))$

Once we have applied the foreground of (1a) to a context (modelled as a record) we will obtain the function (2a) of type (2b)

(2) a. $\lambda r{:}\big[\mathrm{x}{:}\mathit{Ind}\big]\ .\ \big[\mathrm{e}{:}\mathrm{dog}(r.\mathrm{x})\big]$

b. $(\big[\mathrm{x}{:}\mathit{Ind}\big] \rightarrow \mathit{RecType})$

There is a correspondence between this and Montague's treatment of common nouns. Montague (1973) introduces predicates corresponding to common nouns which in his type system are of the type $\langle\langle s, e\rangle, t\rangle$. The type $\langle s, e\rangle$ for Montague is the type of individual concepts. These are modelled as functions from world-time pairs (of type s) to individuals (of type e). The reason that Montague used this type rather than the simpler type $\langle e, t\rangle$, that is, the type of functions from individuals to truth-values, has to do with his treatment of the Partee puzzle concerning temperatures and prices which we will take up below. Much subsequent research has abandoned

Montague's analysis using individual concepts and used the simpler type $\langle e, t\rangle$. This alternative would correspond to (3) in our terms.

(3) a. λx:*Ind* . $\big[$e:dog(x)$\big]$

b. (*Ind* $\rightarrow$ *RecType*)

Ind corresponds to Montague's type e, the type of entities or individuals. Instead of Montague's t, the type of truth values we have *RecType*. Record types serve as our propositions. Thus instead of mapping to a truth value (for Montague, following Frege, the denotation corresponding to a proposition), we map to the proposition itself (see the discussion in Chapter 3, Section 3.4). We will argue that (1) is preferable to (3) in that records which are arguments to such a function are frames and that, among other things, frames as arguments enable us to account for the Partee puzzle. One way of seeing this is that we are using frames to replace Montague's use of individual concepts to approach this problem. In this way our proposal is closely related to work by Löbner (2015). We made this proposal in previous work (Cooper, 2010, 2012b). Here we will present a modification of that proposal which uses frames to introduce scales and measure functions and yields a more general treatment of the semantics of verbs like *rise* than we were able to give in the earlier treatment. In addition it gives us a way of treating nouns like *passenger* where, at least on some readings, we seem to be predicating of passenger events, rather than individual passengers. We will also relate our treatment to other recent work on the introduction of frame semantics into formal semantics.

5.2 The Partee puzzle

The puzzle is one that Barbara Partee raised while sitting in on an early presentation of the material that led to Montague (1973). In its simplest form it is that (4c) should follow from (4a,b) given some otherwise apparently harmless assumptions.

(4) a. The temperature is rising

b. The temperature is ninety

c. Ninety is rising

Clearly, our intuitions are that (4c) does not follow from (4a,b). The assumptions that the error relies on are those given in (5).

(5) a. *temperature* is a predicate of individuals

b. *is* in (4b) represents identity between individuals

Montague's solution was to abandon (5a) and say that 'temperature' is a predicate not of individuals but of individual *concepts*, in his terms functions from world-time pairs to individuals, thus introducing intensionality into predication by common nouns. When we say (4a) we are predicating 'rise' not of an individual but of a function. When we say (4b) we are saying that the *value* of the function at the current world and time is identical with ninety. The technical machinery that Montague uses to achieve this involves his predilection for general treatments. He treats all common nouns as being predicates of individual concepts. But in the case of all nouns other than *price* or *temperature* in his fragment he requires that the individual concepts are rigid designators, that is, they are constant functions which return the same individual for every world-time pair. Similarly intransitive verbs will correspond to predicates of individual concepts but in the case of verbs other than *rise* and *change* (in his fragment) there will be a predicate of the value of the individual concept which holds just in case the verb predicate holds of the individual concept. Finally *be* is treated as representing identity of the values of individual concepts and a given time and world and not identity of the individual concepts. Thus two distinct individual concepts can have identical values at a given world and time.

Given this machinery we can analyze the Partee puzzle represented in (4) as follows. When we say that the temperature is rising we are predicating 'rise' of an individual concept, a function from world-time pairs. Montague does not say what it might mean for such a function to rise. There is, however, something obvious that we could say, namely that if f is such that temperature(f) at world w and time t, then rise(f) is true at world w and time t just in case there is some time t', $t' < t$ ("t' is earlier than t"), and some time t'', $t < t''$, such that $f(w, t')$ is less than $f(w, t'')$. (We may assume that f returns a (real) number for any world and time.) When we say that the temperature f is ninety at world w and time t, what we mean is that $f(w, t) = 90$ (assuming that the interpretation of *ninety* is an individual concept g such that for any world, w, and time, t, $g(w, t) = 90$). From this is does not follow that ninety is rising, that is, rise(g). After all, we have just said that *ninety* corresponds to a constant function which always returns the same value and rising functions have to return different values at different times.

We have now shown that Montague's analysis prevents the offending inference from going through but we must also show that the inference does go through in "normal" cases according to his analysis. Consider (6).

(6) a. The dog is barking
    b. The dog is Fido
    c. Fido is barking

Here we do want (6c) to follow as a conclusion from the premises (6a,b). When we say that the dog is barking we are predicating 'bark' of a constant function f since for an individual concept to fall under the predicate 'dog' it must be rigid, i.e. return the same object for each world and time. Furthermore, there is a predicate, call it 'bark$_*$', such that for any w and t, 'bark$_*$' holds of $f(w,t)$ just in case 'bark' holds of f. So in effect by predicating 'bark' of f at w and t, we are predicating 'bark$_*$' of $f(w,t)$. (Given Montague's notion of proposition, bark(f) and bark$_*$($f(w,t)$) are the *same* proposition since they are true at exactly the same possible worlds and times.) When we say that the dog is Fido at w and t what we mean is that $f(w,t) = g(w,t)$ where g is the individual concept corresponding to *Fido*. According to Montague's theory of proper names g too will be a constant function always returning the same individual, say, 'fido'. Is Fido barking given these assumptions, that is, is bark(g) true at w and t? There are a couple of ways to make the argument. Since both f and g are constant functions if they have the same value at any world and time they will have the same value at all worlds and times, that is, given the classical set theoretic view of functions that Montague is using, f and g will in fact be the same function. Thus if we predicate anything of f it will also hold of g, since they are identical. The other argument involves the nature of the predicate 'bark'. Since bark(f) is equivalent to bark$_*$($f(w,t)$) and, given that $f(w,t) = g(w,t)$, bark$_*$($f(w,t)$) is equivalent to bark$_*$($g(w,t)$), which in turn is equivalent to bark(g), then bark(f) and bark(g) are equivalent. Thus if bark(f) is true, then so is bark(g).

Despite the obvious ingenuity and formal correctness of this solution it fell into disuse. As Löbner (2015) points out one objection is to the interpretation of (4) as an identity statement rather than the location of the temperature value on a scale. This point was made by Jackendoff (1979), a paper which has given rise to a trickle of remarks and replies in *Linguistic Inquiry* over a period of thirty years: Löbner (1981); Lasersohn (2005); Romero (2008). Part of Jackendoff's argument is that in addition to (7a) we can also say (7b), just as we can say (7c).

(7) a. The temperature is ninety
    b. The temperature is at ninety
    c. The airplane is at 6000 feet

We do not, he argues, feel the temptation to conclude (8c) from (8a,b).

(8) a. The airplane is at 6000 feet
    b. The airplane is rising
    c. 6000 feet is/are rising

So neither should we feel the temptation to draw the offending conclusion in the temperature puzzle since even though we say *the temperature is ninety* we mean *the temperature is at ninety*. Jackendoff does not point out, however, that there is an important difference between the temperature and the airplane case, namely that (9) does not mean the same as (8a), and to the extent that it means anything it means something absurd which involves an equality between an airplane and 6000 feet.

(9) The airplane is 6000 feet

If Jackendoff were right that *is* can mean *is at* why would this be the case? Löbner (1981) has a stronger argument against Jackendoff. He points out that we cannot conclude (10c) from (10a,b)

(10) a. The temperature of the air in my refrigerator is the same as the temperature of the air in your refrigerator
b. The temperature of the air in my refrigerator is rising
c. The temperature of the air in your refrigerator is rising

Lasersohn (2005) gives the example in (11) based on Löbner's example.

(11) a. The temperature in Chicago is rising
b. The temperature in Chicago is the very same as the temperature in St. Louis
c. The temperature in St. Louis is rising

These examples are meant to show that there are similar cases to the original Partee puzzle where the construction seems clearly equative rather than locative. Note that we can mention identity explicitly as in (12).

(12) The temperature in Chicago is identical with the temperature in St. Louis

Romero (2008) discusses examples with prices where it seems intuitive that there are two readings, one where the inference does not go through and one where it does.

(13) a. The prices in supermarket A are (the very same as) the prices in supermarket B
b. Most prices in supermarket A are rising
c. Most prices in supermarket B are rising

On one reading (not the preferred one, I think) (13a) means that at the current time the prices just happen to be the same. In this case the inference does not go through. The other reading is that the prices in the two supermarkets are pegged to each other, perhaps because they are owned by the same chain even though they have different names. (Note that this is not quite the same as saying that the prices are *necessarily* the same which is the case that Romero discusses. This is a matter of business strategy, not logic. The supermarket owners *could* have chosen not to peg the prices to each other.) In this case the inference does go through.[1]

Despite all this discussion there is an important intuition in Jackendoff's observation that the interpretation of *the temperature is ninety* involves the placement of the temperature on a scale. In a sense Montague was recognizing this by modelling temperature in terms of his individual concepts. He was giving us a function which returns for each world and time an individual (presumably a number) representing the temperature. Thus he could account for the fact that the temperature is different at different times. The problem is, though, that possible worlds (that is, total ways the universe could be) do not have a single temperature, even at a single point of time. The notion of individual concept he has is simply not fine-grained enough to deal with temperature. One can understand why Montague might not have wanted to pursue this matter further in PTQ. He wanted to include the treatment of temperature in his general treatment of intensions (functions from possible worlds and times to objects of various types) but in order to get temperature right he would have had to change this. One strategy would be to use possible situations (parts of possible worlds). Another strategy would have been to use an additional index, not just worlds and times but also locations. But if he had done this for temperature and maintained a general theory of intensions he would have had to make all intensions be functions defined on triples of worlds, times and locations and this would have raised issues about the relationship between intensionality and indexicality which he was probably wise to avoid at that point in the development. Nevertheless, it is an important issue which nags at some of the central assumptions of formal semantics as Montague was proposing it: namely, the use of possible worlds and evaluation with respect to a finite set of indices some of which are in the domain of intensions and some of which are contextual parameters.

Löbner's early work on this topic (Löbner, 1979, 1981) treated this problem by removing what he called *functional concepts* (*Funktionalbegriffe*) from the general notion of intension and allowing them to have different numbers and types of argument roles. These insights led him in later work (Löbner, 2014, 2015) to adopt a frame semantic approach where the parameters that are relevant for interpretation can vary between different words and phrases and there is no fixed set of indices as there was in the original work on formal semantics. This is very much the same kind of proposal as in Cooper (2010, 2012b) although the historical precursors we had in mind were different. In my case, the precursors were early work on situation semantics such as Barwise and Perry (1983) and frame semantics of the kind suggested in Fillmore (1982, 1985) and taken as a foundation for FrameNet (Ruppenhofer *et al.*, 2006, `https://framenet.icsi.berkeley.edu`). In Löbner's case, the inspiration for frames comes from the psychological work of Barsalou (1992a,b, 1999).

[1] Actually, there is a further complication with these examples involving plural quantifiers, which Romero does not discuss. We also need an assumption that the two supermarkets have sufficiently similar stock. If most of the prices are rising in supermarket A and supermarket B only stocks those items whose prices are not rising in supermarket A, then even though the prices in the two supermarkets are the same (and pegged to each other), the prices in supermarket B are not rising.

5.3 Frames as records

Our leading idea in modelling frames is that they correspond to records and that the *roles* (or *frame elements* in the terminology of FrameNet) are represented by the record fields. Records are in turn what we use to model situations so frames and situations in our view turn out to be the same. Given that we are working in a type theory which makes a clear distinction between types and the objects which belong to those types it is a little unclear whether what we call frame should be a record or a record type. We need both and we will talk of frames (records) and frame types (record types). For example, when we look up the frame Ambient_temperature (`https://framenet2.icsi.berkeley.edu/fnReports/data/frameIndex.xml?frame=Ambient_temperature`) in FrameNet we will take that to be an informal description of a frame type which can be instantiated by the kinds of situations which are described in the examples there. In our terms we can characterize a type corresponding to a very stripped down version of FrameNet's Ambient_temperature which is sufficient for us to make the argument we wish to make. This is the type *AmbTempFrame* defined in (14).

(14) $$\begin{bmatrix} \text{x} & : & \textit{Real} \\ \text{loc} & : & \textit{Loc} \\ \text{e} & : & \text{temp(loc, x)} \end{bmatrix}$$

This is different from the earlier proposal we made in Cooper (2012b) which is given in (15).

(15) $$\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e-time} & : & \textit{Time} \\ \text{e-location} & : & \textit{Loc} \\ \text{c}_{\text{temp\_at\_in}} & : & \text{temp\_at\_in(e-time, e-location, x)} \end{bmatrix}$$

The new proposal in (14) differs from the old one in two ways. Firstly we have removed the field for time. This is because we now want to treat time in terms of strings of events rather than introducing time-points as such. This follows Fernando's strategy (for example in Fernando, 2011) and relates to the discussion of the Russell-Wiener construction of time in Kamp (1979). Secondly we have made the type in the 'x'-field (the field which will contain 'ninety' in our example) be *Real* ("real number") rather than *Ind* ("individual"). As Lasersohn (2005) points out the issue was raised early in the literature as to whether numbers (or temperature measurements at any rate) should be treated as individuals in these examples or should be counted as belonging to a separate type (Bennett, 1974; Thomason, 1979). In our earlier work we assumed that temperatures were to be considered as individuals because we had no reason to do otherwise. In the current analysis, however, we want to build in a notion of scale which involves a mapping to real numbers and therefore we will model temperatures as real numbers. As we will see this will lead to a slight complication in the compositional semantics so there is still an open issue as to whether this is the right decision.

A scale is a function which maps frames (situations) to a real number. Thus a scale for ambient temperature will be of the type (16a) and the obvious function to choose of that type is the function in (16b) which maps any ambient temperature frame to the real number in its ‘x’-field.

(16) a. (*AmbTempFrame* $\rightarrow$ *Real*)

b. λr:*AmbTempFrame* . r.x

Let us call (16b) ζ_{temp}. As a first approximation we can take an event of a temperature rise to be a string of two temperature frames, $r_1 \frown r_2$, where $\zeta_{\text{temp}}(r_1) < \zeta_{\text{temp}}(r_2)$. Using a notation where T^n is the type of strings of length n each of whose members are of type T and where for a given string, s, $s[0]$ is the first member of s, $s[1]$ the second and so on, a first approximation to the type of temperature rises could be (17).

$$(17)\quad \left[\begin{array}{lll} \text{e} & : & \textit{AmbTempFrame}^2 \\ \text{c}_{\text{rise}} & : & \zeta_{\text{temp}}(\text{e}[0]) < \zeta_{\text{temp}}(\text{e}[1]) \end{array}\right]$$

In the c_{rise}-field of (17) we are using $<$ as an infix notation for a predicate ‘less-than’ with arity $\langle$*Real*, *Real*$\rangle$ which obeys the constraint in (18).

(18) less-than(n, m) is non-empty (“true”) iff $n < m$

One way to meet this constraint is to specify the witness conditions for ‘less-than(n, m)’ as in (19).

(19) e : less-than(n, m) iff $n \varepsilon e$, $m \varepsilon e$ and $n < m$

A more general type for temperature rises is given by (20) where we abstract away from the particular temperature scale used by introducing a field for the scale into the record type. This, for example, allows for an event to be a temperature rise independent of whether it is measured on the Fahrenheit or Celsius scales.

$$(20)\quad \left[\begin{array}{lll} \text{scale} & : & (\textit{AmbTempFrame} \rightarrow \textit{Real}) \\ \text{e} & : & \textit{AmbTempFrame}^2 \\ \text{c}_{\text{rise}} & : & \text{scale}(\text{e}[0]) < \text{scale}(\text{e}[1]) \end{array}\right]$$

This type, though, is now too general to count as the type of temperature rising events. To be of this type, it is enough for there to be some scale on which the rise condition holds and the scale is allowed to be any arbitrary function from temperature frames to real numbers. Of course, it is possible to find some arbitrary function which will meet the rise condition even if the temperature is actually going down. For example, consider a function which returns the number on the Celsius scale but with the sign (plus or minus) reversed making temperatures above 0 to be below 0 and *vice versa*. There are two ways we can approach this problem. One is to make the type in the scale-field a subtype of (*AmbTempFrame* → *Real*) which limits the scale to be one of a number of standardly accepted scales. This may be an obvious solution in the case of temperature where it is straightforward to identify the commonly used scales. However, scales are much more generally used in linguistic meaning and people create new scales depending on the situation at hand. This makes it difficult to specify the nature of the relevant scales in advance and we therefore prefer our second way of approaching this problem.

The second way is to parametrize the type of temperature rising events. By this we mean using a dependent type which maps a record providing a scale to a record type modelling the type of temperature rising events according to that scale. The function in (21) is a dependent type which is related in an obvious way to the record type in (20).

(21) $\lambda r{:}\left[\text{scale:}(\textit{AmbTempFrame} \to \textit{Real})\right] .$
$$\left[\begin{array}{lll} \text{e} & : & \textit{AmbTempFrame}^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1]) \end{array}\right]$$

According to (20) an event will be a temperature rise if there is some scale according to which the appropriate relation holds between the temperatures of the two stages of the event which we are comparing. According to (21) on the other hand, there is no absolute type of a temperature rise. We can only say whether an event is a temperature rise with respect to some scale or other. If we choose some non-standard scale like the one that reverses plus and minus temperatures as we suggested above then what we normally call a fall in temperature will in fact be a rise in temperature *according to that scale*. You are in principle allowed to choose whatever scale you like, though if you are using the type in a communicative situation you had better make clear to your interlocutor what scale you are using and perhaps also why you are using this scale as opposed to one of the standardly accepted ones. Like the parametric contents we introduced in Chapter 4, the dependent types introduce a presupposition-like component to communicative situations. We are assuming the existence of some scale in the context.

Why do we characterize the domain of the function in (21) in terms of records containing a scale rather than just scales as in (22)?

(22) $\lambda \sigma{:}(\textit{AmbTempFrame} \to \textit{Real}) .$
$$\left[\begin{array}{lll} \text{e} & : & \textit{AmbTempFrame}^2 \\ \text{c}_{\text{rise}} & : & \sigma(\text{e}[0]) < \sigma(\text{e}[1]) \end{array}\right]$$

The intuitive reason is that we want to think of the arguments to such functions as being contexts, that is situations (frames) modelled as records. The scale will normally be only one of many informational components which can be provided by the context and the use of a record type allows for there to be more components present. In practical terms of developing an analysis it is useful to use a record type to characterize the domain even if we have only isolated one parameter since if further analysis should show that additional parameters are relevant this will mean that we can add fields to the domain type thereby restricting the domain of the function rather than giving it a radically different type.

And indeed in this case we will now show that there is at least one more relevant parameter that needs to be taken account of before we have anything like a reasonable account of the type of temperature rise events. In (14) we specified that an ambient temperature frame relates a real number ("the temperature") to a spatial location. And now we are saying that a temperature rise is a string of two such frames where the temperature is higher in the second frame. But we have not said anything about how the locations in the two frames should be related. For example, suppose I have a string of two temperature frames where the location in the first is London and the location in the second is Marrakesh. Does that constitute a rise in temperature (assuming that the temperature in the second frame is higher than the one in the first)? Certainly not a temperature rise in London, nor in Marrakesh. If you want to talk about a temperature rise in a particular location then both frames have to have that location and we need a way of expressing that restriction. Of course, you can talk about temperature rises which take place as you move from one place to another and which therefore seem to involve distinct locations. However, it seems that even in these cases something has to be kept constant between the two frames. One might analyse it in terms of a constant path to which both locations have to belong or as a constant relative location such as the place where a particular person (or car, or airplane) is. You cannot just pick two arbitrary temperature frames without holding something constant which ties them together. We will deal here with the simple case where the location is kept constant.[2] We will say that the background information for judging an event as a temperature rise has to include not only a scale but also a location which is held constant in the two frames. This is expressed in (23).

(23) $$\lambda r{:}\begin{bmatrix}\text{fix}{:}\begin{bmatrix}\text{loc}{:}\mathit{Loc}\end{bmatrix}\\ \text{scale}{:}(\mathit{AmbTempFrame} \to \mathit{Real})\end{bmatrix}\ .\\ \begin{bmatrix}\text{e} & : & (\mathit{AmbTempFrame} \wedge\!\!\!\!\raisebox{-0.5ex}{.}\ \begin{bmatrix}\text{loc}{=}r.\text{fix.loc}{:}\mathit{Loc}\end{bmatrix})^2\\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1])\end{bmatrix}$$

Here the 'fix'-field in the context is required to be a record which provides a location. One reason for making the 'fix'-field a record rather than simply a location is that we will soon see

[2]Although in astronomical terms, of course, even a location like London is a relative location, that is, where London is according to the rotation of the earth and its orbit around the sun. Thus the simple cases are not really different from the cases apparently involving paths.

an example where more than one parameter needs to be fixed. It will also help us ultimately in characterizing a general type for a rising event (not just a rise in temperature) if we can refer to the type in the 'fix'-field as *Rec* ("record") rather than to list a disjunction of all the various types of the parameters that can be held constant in different cases.

The temperature rise event itself is now required to be a string of two frames which belong to a subtype of *AmbTempFrame*, namely where the 'loc'-field has been made manifest and is specified to have the value specified for 'loc' in the 'fix'-field. Here we are using the record in the 'fix'-field of the argument to the function to partially specify the type *AmbTempFrame* by fixing values for some of its fields. One can think of the 'fix'-record as playing the role of a partial assignment of values to fields in the type. To emphasize this important role and to facilitate making general statements without having to name the particular fields involved, we shall use the operation of specification or anchoring introduced on p. 181ff. This operation maps a record type, T, and a record, r to the result of specifying T with r, which we notate as $T \parallel r$. Using this notation we can rewrite (23) as (24).

(24) $$\lambda r{:}\begin{bmatrix}\text{fix}{:}\begin{bmatrix}\text{loc}{:}\mathit{Loc}\end{bmatrix}\\ \text{scale}{:}(\mathit{AmbTempFrame} \to \mathit{Real})\end{bmatrix} .\\ \begin{bmatrix}\text{e} & : & (\mathit{AmbTempFrame}\|r.\text{fix})^2\\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1])\end{bmatrix}$$

We will call (24) *TempRiseEvent* and its domain type *TempRiseEventCntxt*. This means that given a record, c, of type *TempRiseEventCntxt* we can make a judgement such as that given in (25).

(25) e : *TempRiseEvent*(c)

That is, we judge that e is a temperature rising event according to the context c.

This is still a very simple theory of what a temperature rise event may be but it will be sufficient for our current purposes. How might we use this to specify a content for the intransitive verb *rise* in a sentence like *the temperature is rising*? First we define a predicated 'rise' which takes two arguments which are both records. Thus the arity of 'rise' is $\langle$*Rec*, *Rec*$\rangle$. The first record can be an ambient temperature frame and the second a temperature rise event context. We specify a witness condition associated with 'rise' in (26).

(26) e : rise(r, c) if

r : *AmbTempFrame*,
c : *TempRiseEventCntxt* and

$$e : \textit{TempRiseEvent}(c) \mathbin{\dot\wedge} \left[\text{e}:\left[\text{t}_0{=}r{:}\textit{AmbTempFrame}\right]\right]$$

Note first that in (26) we use ‘if’ rather than ‘iff’. This is not the only witness condition which we will associate with ‘rise’; it represents a sufficient but not necessary condition. Note also that the third condition specifies that e is a temperature rise event and that the first item in the string of two ambient temperature frames thus specified is the temperature frame r, that is, the first argument to the predicate. (Recall that a string e_1e_2 is modelled as the record in (27).)

(27) $\begin{bmatrix} \text{t}_0 & = & e_1 \\ \text{t}_1 & = & e_2 \end{bmatrix}$

The intuition is, then, that when we predicate ‘rise’ of an ambient temperature frame, we are saying that it is the initial frame in a temperature rising event. We can use this predicate in the characterization of a parametric content for the verb *rise*, given in (28).

(28) $\ulcorner \lambda c{:}\begin{bmatrix} \textit{Cntxt} \\ \mathfrak{c}{:}\textit{TempRiseEventCntxt} \end{bmatrix} .\ \lambda r{:}\left[\text{x}{:}\textit{Rec}\right] .\ \left[\ \text{e} \ : \ \text{rise}(r, c.\mathfrak{c})\ \right]\urcorner$

However, the verb *rise* can be used to talk about other kinds of rising events than temperature rises and we will need different parametric contents for other cases.

We move on now to price rise events. We will take (29) to be the type of price frames, *Price-Frame*.

(29) $\begin{bmatrix} \text{x} & : & \textit{Real} \\ \text{loc} & : & \textit{Loc} \\ \text{commodity} & : & \textit{Ind} \\ \text{e} & : & \text{price(commodity, loc, x)} \end{bmatrix}$

The fields represented here are based on a much stripped down version of the FrameNet frame `Commerce_scenario` where our ‘commdodity’-field corresponds to the frame element called ‘goods’ and the ‘x’-field corresponds to the frame element ‘money’. A price rise is a string of two price frames where the value in the ‘x’-field is higher in the second. Here, as in the case of a temperature rise, we need to keep the location constant. It does not make sense to say that a price rise has taken place if we compare a price in Marrakesh with a price in London, even though the price in London may be higher. In the case of price we also need to keep the commodity constant, something that does not figure at all in ambient temperature. We cannot say that a price rise has taken place if we have the price of tomatoes in the first frame and the price of oranges in the second frame. Thus, following the model of (24), we can characterize the dependent type of price rises as (30).

(30) λr:$\begin{bmatrix}\text{fix:}\begin{bmatrix}\text{loc:}\textit{Loc}\\ \text{commodity:}\textit{Ind}\end{bmatrix}\\ \text{scale:}(\textit{PriceFrame} \rightarrow \textit{Real})\end{bmatrix}$.
$\begin{bmatrix}\text{e} & : & (\textit{PriceFrame}\|r.\text{fix})^2\\ \text{c}_{\text{rise}} & : & r.\text{scale(e[0])} < r.\text{scale(e[1])}\end{bmatrix}$

We call (30) *PriceRiseEvent* and its domain type *PriceRiseEventCntxt*. We can add a new witness condition associated with 'rise'.

(31) e : rise(r, c) if

r : *PriceFrame*,
c : *PriceRiseEventCntxt* and
e : *PriceRiseEvent*(c) $\dot{\wedge}$ $\begin{bmatrix}\text{e:}\begin{bmatrix}\text{t}_0\text{=}r\text{:}\textit{PriceFrame}\end{bmatrix}\end{bmatrix}$

We can construct a parametric content for the verb *rise* which exploits this witness condition. This is given in (32).

(32) $\ulcorner\lambda c$:$\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{c}\text{:}\textit{PriceRiseEventCntxt}\end{bmatrix}$. λr:$\begin{bmatrix}\text{x:}\textit{Rec}\end{bmatrix}$. $\begin{bmatrix}\text{e} & : & \text{rise}(r, c.\mathfrak{c})\end{bmatrix}\urcorner$

Finally we consider a third kind of rising event discussed in Cooper (2012b) based on the example in (33).

(33) As they get to deck, they see the Inquisitor, calling out to a Titan in the seas. The giant Titan rises through the waves, shrieking at the Inquisitor.

`http://en.wikipedia.org/wiki/Risen_(video_game)`
accessed 4th February, 2010

Here what needs to be kept constant in the rising event is the Titan. What needs to change between the two frames in the event is the height of the location of the Titan. Thus in this example the location is *not* kept constant. In order to analyze this we can use location frames of the type *LocFrame* as given in (34).

(34) $\begin{bmatrix}\text{x} & : & \textit{Ind}\\ \text{loc} & : & \textit{Loc}\\ \text{e} & : & \text{at(x, loc)}\end{bmatrix}$

The dependent type, *LocRiseEvent*, with domain type *LocRiseEventCntxt*, for a rise in location event is (35).

(35) $\lambda r{:}\begin{bmatrix}\text{fix:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix}\\ \text{scale:}(\mathit{LocFrame} \to \mathit{Real})\end{bmatrix}\ .$
$\begin{bmatrix}\text{e} & : & (\mathit{LocFrame}\|r.\text{fix})^2\\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1])\end{bmatrix}$

Here the obvious scale function does not simply return the value of a field in the location frame. What is needed is a scale based on the height of the location. One way to do this would be to characterize the type of locations, *Loc*, as the type of points in three-dimensional Euclidean space. That is, we consider *Loc* to be an abbreviation for (36).

(36) $\begin{bmatrix}\text{x-coord} & : & \mathit{Real}\\ \text{y-coord} & : & \mathit{Real}\\ \text{z-coord} & : & \mathit{Real}\end{bmatrix}$

Each of the fields in (36) corresponds to a coordinate in Euclidean space. A more adequate treatment would be to consider locations as regions in Euclidean space but we will not pursue that here. Treating *Loc* as (36) means that we can characterize the scale function, ζ_{height}, as returning the height of the location in the location frame, as in (37).

(37) $\lambda r{:}\mathit{LocFrame}\ .\ r.\text{loc.z-coord}$

If we wish to restrict the dependent type of rising events to vertical rises we can fix the x and y-coordinates of the location as in (38).

(38) $\lambda r{:}\begin{bmatrix}\text{fix:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{loc:}\begin{bmatrix}\text{x-coord:}\mathit{Real}\\ \text{y-coord:}\mathit{Real}\end{bmatrix}\end{bmatrix}\\ \text{scale:}(\mathit{LocFrame} \to \mathit{Real})\end{bmatrix}\ .$
$\begin{bmatrix}\text{e} & : & (\mathit{LocFrame}\|r.\text{fix})^2\\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1])\end{bmatrix}$

We can now add a new witness condition associated with ‘rise’, given in (39).

(39) e : rise(r, c) if

r : *LocFrame*,
c : *LocRiseEventCntxt* and
e : *LocRiseEvent*(c) $\dot{\wedge}$ $\left[\text{e}:\left[\text{t}_0{=}r{:}\textit{LocFrame}\right]\right]$

We can use this predicate to create a parametric content for the intransitive verb *rise*, as in (40).

$$\text{(40)}\quad \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{c}{:}\textit{LocRiseEventCntxt}\end{bmatrix} .\ \lambda r{:}\left[\text{x}{:}\textit{Rec}\right] .\ \left[\ \text{e} \quad : \quad \text{rise}(r.\text{x},\, c.\mathfrak{c})\ \right]\urcorner$$

We have now characterized three kinds of rising events. In Cooper (2010, 2012b) we argued that there is in principle no limit to the different kinds of rising events which can be referred to in natural language and that new types are created on the fly as the need arises. The formulation in those works did not allow us to express what all these particular meanings have in common. We were only able to say that the various meanings seem to have some kind of family resemblance. Now that we have abstracted out scales and parameters to be fixed we have an opportunity to formulate something more general. There are two things that vary across the different dependent types that we have characterized for risings. One is the frame type being considered and the other is the type of the record which contains the parameters held constant in the rising event. If we abstract over both of these we have a characterization of rising events in general. This is given in (41).

$$\text{(41)}\quad \lambda r{:}\begin{bmatrix}\text{frame\_type}{:}\textit{RecType}\\ \text{fix\_type}{:}\textit{RecType}\\ \text{fix}{:}\text{fix\_type}\\ \text{scale}{:}(\text{frame\_type} \rightarrow \textit{Real})\end{bmatrix} .$$
$$\begin{bmatrix}\text{e} & : & (r.\text{frame\_type}\|r.\text{fix})^2\\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1])\end{bmatrix}$$

(41) is so general (virtually everything of content has been parametrized) that it may be hard to see it as being used in the characterization of the meaning of *rise*. What seems important for characterizing the meanings of *rise* that a speaker has acquired is precisely the collection of frame types, and associated fix types and scales which an agent has developed through experience. (41) seems to be relevant to a kind of meta-meaning which specifies what kind of contents can be associated with the word *rise*. In this sense it seems related to the notion of *meaning potential*, a term which has its origins in the work of Halliday (1977) where meanings are spoken of informally as being "created by the social system" and charaterized as "integrated systems of meaning potential" (p. 199). The notion is much discussed in more recent literature, for example, Linell (2009), where meaning potential is discussed in the following terms: "Lexical meaning potentials are (partly) open meaning resources, where actual meanings can only emerge in specific, situated interactions" (p. 330). The parametric contents for *rise* that we have presented

here (and included in the summary of resources in Section 5.8) are examples of what Linell is calling here "actual meanings".

5.4 Frames and common nouns

A central aspect of our analysis of the Partee puzzle is that the contents of common nouns are functions that take frames, that is records, as arguments. Nevertheless, we make a distinction between individual level predicates like 'dog' whose arity is $\langle \mathit{Ind} \rangle$ and frame level predicates like 'temperature' whose arity is $\langle \mathit{Rec} \rangle$. Leaving aside for now the need for parametric contents, the content for associated with an utterance event of type "dog" would be (2a) repeated here as (42a). This is contrasted with the content for an utterance of type "temperature" given in (42b).

(42) a. $\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right] \,.\, \left[\ \text{e} \ : \ \text{dog}(r.\text{x})\ \right]$

b. $\lambda r{:}\left[\text{x}{:}\mathit{Rec}\right] \,.\, \left[\ \text{e} \ : \ \text{temperature}(r.\text{x})\ \right]$

We make an exactly similar distinction between individual level and frame level verb phrases. In (43) we present contents which can be associated with utterances of type "run" and "rise" respectively.

(43) a. $\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right] \,.\, \left[\ \text{e} \ : \ \text{run}(r.\text{x})\ \right]$

b. $\lambda r{:}\left[\text{x}{:}\mathit{Rec}\right] \,.\, \left[\ \text{e} \ : \ \text{rise}(r.\text{x})\ \right]$

The types which we associate with the individual level and frame level properties in (42) and (43) are given in (44).

(44) a. $(\left[\text{x}{:}\mathit{Ind}\right] \rightarrow \mathit{RecType})$

b. $(\left[\text{x}{:}\mathit{Rec}\right] \rightarrow \mathit{RecType})$

While these types are distinct, they are nevertheless related in that they both have the same range type and the domain types of (44a) and (44b) are both record types requiring a field with the label 'x'. Up until now we have used *Ppty* ("property") to designate (44a). Now we might be more specific and designate it as *IndPpty* ("property of individuals") and use *FramePpty* ("property of frames") to designate (44b).

Given that TTR has join (disjunctive) types (Appendix 7) we always have the option of forming the join of those types which we want to represent types of properties. Thus, given the two types of properties we have seen so far we can form the join type in (45).

(45) $(([\text{x}:\textit{Ind}] \rightarrow \textit{RecType}) \vee ([\text{x}:\textit{Rec}] \rightarrow \textit{RecType}))$

If there are more types of properties we wish to add to the general type of properties we can form a larger join type to include them. We can always form a join type based on any finite collection of types. Using join types in this way we can create a type which has all the witnesses of any finite collection of types. We cannot, however, express a type corresponding to an infinite set of types in this way. In addition using join types in this way does not make explicit any relationship between the various types in the collection, in this case that all the types are function types whose range type is *RecType* and whose domain type is a record type with an 'x'-field. In order to deal with this kind of case, we will use the same technique as we used for parametric contents in Chapter 4. We will first define the type *xType* as a basic type of record types that have a 'x' among its labels, as specified in (46).

(46) T : *xType* iff T : *RecType* and x $\in \text{labels}(T)$

We will treat properties as a pair (that is, a record with two fields) consisting of a type (labelled with 'bg') and a function (labelled with 'fg') corresponding to what we have up to now been calling a property. (47a) is an example of the new kind of property which we normally represent as (47b) and (47c) is the new definition of the type, *Ppty*, of properties.[3]

(47) a. $\begin{bmatrix} \text{bg} & = & [\text{x}:\textit{Ind}] \\ \text{fg} & = & \lambda r{:}[\text{x}:\textit{Ind}]\ .\ [\text{e}:\text{dog}(r.\text{x})] \end{bmatrix}$

b. $\ulcorner \lambda r{:}[\text{x}:\textit{Ind}]\ .\ [\text{e}:\text{dog}(r.\text{x})] \urcorner$

c. $\begin{bmatrix} \text{bg} & : & \textit{xType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{RecType}) \end{bmatrix}$

If P is a property as in (47a,b), we will use $P(r)$ to represent $P.\text{fg}(r)$ (the result of applying the function $P.\text{fg}$ to r).

Actually, things are a little more complicated than this. Using this polymorphic type for properties introduces a complication when we define functions on properties. This is illustrated in the example in (48) where we assume that a:*Ind*, p is a predicate whose arity is $\langle$*Rec*$\rangle$ and that the types *Ind* and *Rec* preclude each other, that is, nothing can be of both type *Ind* and type *Rec*.

[3]For a similar kind of case, though a different approach to treating it, see the discussion in Ginzburg *et al.* (2014), p. 93, of the type of Austinian questions. We have also treated this kind of case in terms of a limited kind of polymorphism, for example, in Ginzburg and Cooper (2014).

(48) a. $\lambda P{:}\textit{Ppty}\;.\;P(\left[\text{x}{=}a\right])\,(\ulcorner\lambda r{:}\left[\text{x}{:}\textit{Rec}\right]\;.\;\left[\;\text{e}\;:\;p(r.\text{x})\;\right]\urcorner)$

b. $\ulcorner\lambda r{:}\left[\text{x}{:}\textit{Rec}\right]\;.\;\left[\;\text{e}\;:\;p(r.\text{x})\;\right]\urcorner\,(\left[\text{x}{=}a\right])$

In (48a) the argument $\ulcorner\lambda r{:}\left[\text{x}{:}\textit{Rec}\right]\;.\;\left[\;\text{e}\;:\;p(r.\text{x})\;\right]\urcorner$ seems appropriate for the function $\lambda P{:}\textit{Ppty}\;.\;P(\left[\text{x}{=}a\right])$ since according to our new definition the argument is indeed of type *Ppty*. However, it appears that the result of β-conversion given in (48b) leads to a situation where the argument $\left[\text{x}{:}a\right]$ is not appropriate to the function $\lambda r{:}\left[\text{x}{:}\textit{Rec}\right]\;.\;\left[\;\text{e}\;:\;p(r.\text{x})\;\right]$ because a is an individual and not a record. This appears to lead to a contradictory situation where (48a) is both well-formed and ill-formed because by β-conversion (48a and b) represent the same object. We avoid this problem by saying that the application of a property, P, to an argument, a, if P.fg(a) in the case where $a : P$.bg but that otherwise it returns a type that is necessarily empty. We introduce a distinguished type, '$\bot$', called "bottom", which is necessarily empty. We can achieve this, for example, by giving it the witness condition in (49).

(49) $a : \bot$ iff $a \neq a$

We can then define property application more accurately than we did above as in (50).

(50) If P : *Ppty* and a is some object, then

1. $P(a) = P.\text{fg}(a)$ if $a : P.\text{bg}$
2. otherwise $P(a) = \bot$

An advantage of this approach to properties is that providing inappropriate arguments to properties will result in something that is necessarily false but not ill-formed while its negation will also be well-formed and true. Thus while (51a) seems strange because individual numbers do not rise (51b) seems true for the same reason.

(51) a. 90 rises

b. 90 does not rise

Now that we have generalized our definition of property in this way, we need also to adjust our definition of property extension. The definition we gave in Chapter 3, example (45), is repeated in (52).[4]

[4] Recall that the notation $\lceil T\rceil$ is defined by

(52) $[\downarrow P] = \{a \mid \exists r[r : \left[\text{x}:\textit{Ind}\right] \wedge r.\text{x} = a \wedge [\check{}P(r)] \neq \emptyset]\}$

This definition is based on the assumption that all properties are of type ($\left[\text{x}:\textit{Ind}\right] \rightarrow$ *RecType*). Now we need to modify it so that we will have a notion of property extension for our new definition of *Ppty*. This is done in (53).

(53) If T : *xType* and P : ($T \rightarrow$ *RecType*) then $[\downarrow P] = \{a \mid \exists r[r : T \wedge r.\text{x} = a \wedge [\check{}P(r)] \neq \emptyset]\}$

In characterizing the content of the noun *temperature* we used the predicate 'temperature'. This predicate has the arity $\langle$*Rec*$\rangle$. What is the relationship between this predicate and the type *AmbTempFrame*, which is a type of records. An intuition that seems reasonable if we are treating *temperature* as a frame level noun is that one kind of frame that could count as a temperature is an ambient temperature frame. We can express this by the witness condition in (54).

(54) s : temperature(r) if r : *AmbTempFrame* and $s = r$

This witness condition corresponds to the inference rule in (55).

(55) $$\frac{r : \textit{AmbTempFrame}}{r : \text{temperature}(r)}$$

Note that in (54) we again use *if* rather than *iff*. This represents one way in which a record can be regarded as a temperature. This allows for there to be other ways to be regarded as a temperature. Note also that we are talking of judgements of the form $a : p(a)$ where the witness for the type is identical with the argument to the predicate. This might appear to be introducing some kind of non-wellfoundedness. However, this is not the case. We can think of the type $p(a)$ in such a case as corresponding to a singleton type T_a, that is, it is either witnessed by a or nothing.

When we say *the temperature is rising* we are talking about an event which is a temperature rise, not a price rise or any other kind of rise. Somehow we have to coordinate the frame which is chosen in connection with the interpretation of temperature with the frame which is chosen in connection with the interpretation of *rise*. The solution to this that we wish to propose rests on the treatment of generalized quantifiers proposed in Cooper (2011, 2013a).

$[\check{T}] = \{a \mid a : T\}$

5.5 Definite descriptions as dynamic generalized quantifiers

In Chapter 3 we showed how to treat indefinite descriptions (consisting of an indefinite article and a common noun phrase) as generalized quantifiers. We will now do something similar for definite descriptions (consisting of a definite article and a common noun phrase). We will then show how to modify this static interpretation of generalized quantifiers so that it becomes a dynamic treatment as presented in Cooper (2011). We will see that the dynamic treatment accounts for how the frame associated with the noun is passed to the verb.

We will treat the definite article *the* as introducing a uniqueness condition. We say that a property is unique in a situation just in case its property extension in that situation is a singleton set. We will make this precise by introducing a predicate 'unique' whose arity is $\langle$*Ppty*$\rangle$. We characterize a witness condition associated with this predicate using the notion of restricted type introduced in Chapter 3, pp. 127*ff*, but adjusted to take account of the new kind of property as in (56).

(56) If P is a property, then $P\restriction r$ is

$$\begin{bmatrix} \text{bg} & = & P.\text{bg} \\ \text{fg} & = & P.\text{fg}\restriction r \end{bmatrix}$$

The witness condition for 'unique' is given in (57).

(57) If P:*Ppty* and s:*Rec*,

then s : unique(P) iff $|\,[{\downarrow}P\restriction s]\,| = 1$

(57) says that a situation, s, is of the type 'unique(P)' (where P is a property) just in case there is exactly one object which has the property P in some component of s. We could introduce a generalized quantifier predicate, 'the', with arity $\langle$*Ppty*, *Ppty*$\rangle$, associated with the witness condition in (58).

(58) If P, Q:*Ppty* then,

$s : \mathrm{the}(P, Q)$ iff s : unique(P) and $[{\downarrow}P\restriction s] \subseteq [{\downarrow}Q\restriction s]$

This corresponds to the Russellian definite description analysis as used by Montague (see Dowty *et al.*, 1981 for an explanation of this) except that here the uniqueness is restricted to a particular situation. This means that we could talk about "the dog" without requiring that there is exactly one dog in the universe.

It is well-known that the uniqueness condition in the Russellian treatment of definite descriptions used by Montague is not quite right for natural language. (For a detailed discussion of the issues involved see Elbourne, 2012.) We can, for example, use the definite description *the dog* even though there are several dogs. It is not a simple matter of restricting ourselves to a particular situation that we are describing since we may be describing a situation with several dogs but still refer to some particular dog in the situation as *the dog*. Such examples are discussed in Cooper (1996) citing (59) from McCawley (1979).

(59) The dog had a fight with another dog yesterday

Our solution to this is to in effect introduce resource situations (Barwise and Perry, 1983; Cooper, 1996). (A similar proposal is made by Elbourne, 2012.) We follow the analysis in Cooper (2013b) and exploit the fact that properties can be restricted to a particular situation by introducing a restricted field in the foreground as in (60).

(60) $\ulcorner \lambda r{:}\big[\text{x:}\mathit{Ind}\big].\ \big[\text{e}{\sqsubseteq}s\text{:dog}(r.\text{x})\big]\urcorner$

For the restricted field notation $\big[\text{e}{\sqsubseteq}s\text{:dog}(r.\text{x})\big]$ see Chapter 3, p. 127. (60) can be glossed as "the property of being a dog in s". We will abbreviate this as 'dog$'{\restriction}s$' where we use 'dog$'$' to abbreviate the property without the restricted field. These abbreviations are represented in (61).

(61) a. dog$'$ abbreviates $\ulcorner \lambda r{:}\big[\text{x:}\mathit{Ind}\big].\ \big[\text{e:dog}(r.\text{x})\big]\urcorner$
b. dog$'{\restriction}s$ abbreviates $\ulcorner \lambda r{:}\big[\text{x:}\mathit{Ind}\big].\ \big[\text{e}{\sqsubseteq}s\text{:dog}(r.\text{x})\big]\urcorner$

In general we can say that if p is a predicate with arity $\langle T\rangle$ then p' represents (62)

(62) $\ulcorner \lambda r{:}\big[\text{x:}T\big]\ .\ \big[\ \text{e} \ :\ p(r.\text{x})\ \big]\urcorner$

We could introduce a function 'SemDefArt' on the model of 'SemIndefArt' which was defined in Chapter 3, example (36), although modified to accommodate parametric contents. This is given in (63).

(63) $\ulcorner \lambda c{:}\mathit{Cntxt}$.
$\lambda Q{:}\mathit{Ppty}$.

$$\lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q & : & \mathit{Ppty} \\ \text{scope}{=}P & : & \mathit{Ppty} \\ \text{e} & : & \text{the(restr, scope)} \end{bmatrix}\urcorner$$

The reason that we need a uniqueness predicate of this kind has to do with the nature of our type theory. The typing mechanism allows us to say for example what is given in (64).

(64) $s : \text{dog}(a)$

One way to paraphrase (64) is "a is a dog in s". It says that s is of type 'dog(a)' but does not rule out that s can be of other types as well including possibly 'dog(b)' where b is distinct from a. We do not have a way of saying that 'dog(a)' is the only type to which s belongs. This would correspond to Schubert's (2000) notion of characterizing a situation, that is, in our terms, presenting an exhaustive list of types to which it belongs, which given that we have meet types (Appendix 8), corresponds to providing a single type to which it belongs such that there is no other type to which it belongs. We have made this choice because it would be very hard if not impossible to guarantee that anything belongs to just one type in the kind of type system we have introduced. Consider, for example, join types. Given our definition of join types in Appendix 7 if any object a is of some type T it will also be of type $(T \vee T')$ for any type T'. Introduction of this classical kind of disjunction into the system makes it difficult to define a useful notion of a type that completely characterizes an object or situation in the way that Schubert wants.[5]

Introducing the predicate 'unique' in the way that we have allows us to place a constraint on the types to which a situation belongs without having to give a complete characterization of all the types to which it belongs. Defining it as a predicate whose argument is a property means that its argument, the property, involves a type. A property is a function which returns a type. Technically, we call it a dependent type. In Chapter 6 we will suggest that allowing types or dependent types as arguments to predicates is a characteristic of evolutionary higher organisms (at least humans). It seems intuitive that the kind of uniqueness involved in the semantics of definite descriptions should belong to this higher kind of reasoning. We can imagine simple organisms (perhaps even as simple as an amoeba) which respond to situations of certain types in certain ways, for example, eating behaviour when confronted with a situation in which an item of food is present. However, it seems unintuitive that such a simple organism would be programmed to engage in eating behaviour when exactly one item of food is present and not otherwise.

[5]Schubert's argument for needing the notion of characterization has to do with defining a causation relation between events. It seems to me that an analysis of causality must involve a type of the causing event. Thus in addition to a two-place cause relation between two events, "e_1 caused e_2", we need a three-place cause relation between two events and a type of the first event, "e_1 caused e_2 in virtue of the fact that $e_1 : T$". Thus, to take an example that Schubert discusses, *John's singing in the shower caused Mary to wake up in virtue of the fact that it was a singing event* but not *John's singing in the shower caused Mary to wake up in virtue of the fact that it was an event in the shower*. Allowing types to be arguments to predicates in the way that we do provides a different solution to the problem that Schubert presents.

Despite the success of our analysis so far in reducing the uniqueness condition to a particular situation rather than applying it to the whole universe, there is another problem with the use of Russellian definite descriptions which it does not address. This is that the uniqueness is treated as part of the assertion, that is, as something that is at issue. This means that we should be able to deny an utterance of a sentence like (65a) or answer the question (65b) with something like (65c).

(65) a. The dog barked.
b. Did the dog bark?
c. No, we don't have a dog.

If (65c) is at all a possible response to (65a or b) then it feels like the denial of a presupposition. When a definite description is used in dialogue it seems that there is an assumption that the interlocutor will be able to identify a relevant situation in which there is a unique dog in a similar way as is suggested in our proposal for the identification of referents of uses of proper names in Chapter 4, that is, looking first on the dialogue gameboard, then in long term memory and if nothing is to be found then accommodating a situation in which there is a unique dog.

We shall use uniqueness to create a presuppositional account of definite descriptions using the techniques for parametric contents which we developed in Chapter 4. The presupposition type (a version of that proposed in Cooper, 2013b adjusted to the new one-place predicate 'unique') is given in (66).

(66) $\left[\text{e:unique(dog}')\right]$

This is the type that, according to the techniques developed in Chapter 4, will need to be matched against an agent's resources (gameboard or long term memory) or, if a match is not available, will need to be accommodated into the agent's gameboard. It requires there to be a situation which has exactly one dog in it. Satisfying the uniqueness presupposition on this view is not so much a question of determining the way the world is (i.e. whether the dog is in some objective sense unique) as determining how the agent has carved up the world into situations.

(66) will, then, be the background of the parametric content of the noun-phrase *the dog*. Three different options for this parametric content present themselves, as in (67).

(67) a. $$\ulcorner \lambda c{:}\begin{bmatrix} \textit{Cntxt} \\ \mathfrak{c}{:}\left[\text{e:unique(dog}')\right] \end{bmatrix} . \\ \lambda P{:}\textit{Ppty} \,.\, \begin{bmatrix} \text{restr=dog}'{\upharpoonright}c.\mathfrak{c}.\text{e} & : & \textit{Ppty} \\ \text{scope=}P & : & \textit{Ppty} \\ \text{e} & : & \text{the(restr, scope)} \end{bmatrix} \urcorner$$

b. $$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{e:unique}(\text{dog}')\end{bmatrix}\end{bmatrix}\,.\;\lambda P{:}\mathit{Ppty}\,.\begin{bmatrix}\text{restr=dog}'{\restriction}c.\mathfrak{c}.\text{e} & : & \mathit{Ppty}\\ \text{scope=}P & : & \mathit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{bmatrix}\urcorner$$

c. $$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{e:unique}(\text{dog}')\end{bmatrix}\end{bmatrix}\,.\;\lambda P{:}\mathit{Ppty}\,.\begin{bmatrix}\text{restr=dog}'{\restriction}c.\mathfrak{c}.\text{e} & : & \mathit{Ppty}\\ \text{scope=}P & : & \mathit{Ppty}\\ \text{e} & : & \text{every(restr, scope)}\end{bmatrix}\urcorner$$

It does not make much difference which of these you choose for the analysis of singular definite descriptions. In Cooper (2013b) we chose the option corresponding to (67c), which offers some vague hope of being able to draw a parallel with plural definites. Note that choosing (67b) or (67c) eliminates the need for the predicate 'the'. Here we will choose (67c). This assumes that we have a predicate 'every' with arity $\langle \mathit{Ppty}, \mathit{Ppty}\rangle$ with the witness condition (68).

(68) s : every(P,Q) iff $[{\downarrow}P] \subseteq [{\downarrow}Q{\restriction}s]$

From the perspective of compositional semantics it is important that the common noun *dog* in these examples plays a role twice in the parametric content of the noun phrase *the dog*: once in determining the type of the context and once as the first argument to the quantifier. In order to achieve the contribution to the context we will need to treat the content of *the* not as a parametric content but as a function from properties (corresponding to the common noun) to a parametric content. The content of *the*, 'SemDefArt', is given in (69).

(69) $$\lambda Q{:}\mathit{Ppty}\,.\;\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{e:unique}(Q)\end{bmatrix}\end{bmatrix}\,.\;\lambda P{:}\mathit{Ppty}\,.\begin{bmatrix}\text{restr=}Q{\restriction}c.\mathfrak{c}.\text{e} & : & \mathit{Ppty}\\ \text{scope=}P & : & \mathit{Ppty}\\ \text{e} & : & \text{every(restr, scope)}\end{bmatrix}\urcorner$$

For the sake of consistency in how determiners are combined with nouns we shall adjust the definition of the 'SemIndefArt' so that it too is a dependent parametric content of the same form as 'SemDefArt' even though it does not introduce a presupposition that depends on the following noun. 'SemIndefArt' is defined as (70).

(70) $\lambda Q{:}\mathit{Ppty}\ .$
$$\ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda P{:}\mathit{Ppty}\ .\left[\begin{array}{lll}\text{restr}{=}Q & : & \mathit{Ppty}\\ \text{scope}{=}P & : & \mathit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{array}\right]\urcorner$$

How should these contents be combined with the content of a common noun to form the content of the noun-phrase? Let us refer to (70) as **the** and use **dog** to refer to the parametric content associated with *dog* given in (71), where 'dog′' represents the property of being a dog as we defined earlier.

(71) $\lambda c{:}\mathit{Cntxt}\ .\ \text{dog}'$

(This assumes that the content for *dog* does not depend on the context.) We can derive a parametric content for *the dog* as indicated in (72).

(72) $$\lambda c{:}\left[\begin{array}{lll}\mathit{Cntxt} & & \\ \mathfrak{c} & : & \left[\begin{array}{lll}\text{s} & : & \textbf{dog}.\text{bg}\\ \text{f} & : & \textbf{the}(\textbf{dog}(\text{s})).\text{bg}\\ \text{a}{=}\text{s}.\mathfrak{c} & : & \mathit{PropCntxt}\end{array}\right]\\ \mathfrak{s}{=}\mathfrak{c}.\text{s}.\mathfrak{s} & : & \mathit{Assgnmnt}\end{array}\right]\ .\ [\textbf{the}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}([\textbf{dog}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}(c))(c)$$

In order to achieve this we need a variant of the operation 'ContForwardApp' defined on p. 173 which combines contents of phrases based on forward application. We will call new variant 'ContForwardApp$_{@@}$' since it makes use of a combination operation we designate by '@@' whereas the normal version uses the combination operation '@', whose current version is defined on p.204. 'ContForwardApp$_{@@}$' is given in (73a) and '@@' is defined in (73b).

(73) a. If T_{arg} and T_{res} are types, then ContForwardApp$_{@@}(T_{\text{arg}}, T_{\text{res}})$ is
$$\lambda u{:}\left[\text{cont:}(T_{\text{arg}}\to T_{\text{res}})\right]{}^\frown\left[\text{cont:}T_{\text{arg}}\right]\ .\ \left[\text{cont}{=}u[0].\text{cont}@@u[1].\text{cont:}T_{\text{res}}\right]$$

b. If $\alpha : (T_1 \to \left[\begin{array}{l}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\to T_2)\end{array}\right])$ and $\beta : \left[\begin{array}{l}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\to T_1)\end{array}\right]$ then the *combination of* α *and* β *based on functional application*, $\alpha@@\beta$, is

$$\ulcorner\lambda c{:}\left[\begin{array}{lll} \textit{Cntxt} & & \\ \mathfrak{c} & : & \left[\begin{array}{lll} \text{s} & : & \beta.\text{bg} \\ \text{f} & : & \alpha(\beta(s)).\text{bg} \\ \text{a=s.}\mathfrak{c} & : & \textit{PropCntxt} \end{array}\right] \\ \mathfrak{s}\text{=}\mathfrak{c}\text{.s.}\mathfrak{s} & : & \textit{Assgnmnt} \end{array}\right] .$$

$$[\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}([\beta]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}(c))(c)\urcorner$$

What we have characterized so far is a static treatment of generalized quantifiers. Dynamic generalized quantifiers as presented in Cooper (2011) involving changing the constraint on the quantifier predicate so that the information represented by the first argument to the quantifier predicate is passed on as a restriction to the second argument of the predicate. What we mean by the information associated with a property P is essentially the merge of the domain type and the range type of the dependent type which is the foreground of the property. Consider the example ‘dog′’ repeated in (74).

(74) $\lambda r{:}\left[\text{x:}\textit{Ind}\right] . \left[\begin{array}{lll} \text{e} & : & \text{dog}(r.\text{x}) \end{array}\right]$

Informally the information associated with this which is to be passed on to the second argument of the quantifier is that we are talking about some individual which is a dog. This is represented by the record type in (75).

(75) $\left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \end{array}\right]$

Intuitively, the type in (75) can be obtained from ‘dog′’ by merging the domain type with the result of relabelling the range type so that ‘r.x’ is replaced by ‘x’, that is, (76).

(76) $\left[\begin{array}{lll} \text{e} & : & \text{dog(x)} \end{array}\right]$

We call (75) a *fixed point type* for ‘dog′’ because for any record r of this type, r : dog′(r). We use ‘$\mathcal{F}$(dog′)’ to designate (75).

There are some complications which need to be noted concerning fixed point types. Not all functions which return record types will have fixed points. For example, if the domain type of some function, $\mathcal{T}$ requires a non-record, and $\mathcal{T}$ returns a record type, then there cannot be any a such that $a : \mathcal{T}(a)$.

Even if $\mathcal{T}$ maps records to record types, it will not always be the case that $\mathcal{T}$ has fixed points. Consider (77).

(77) $\lambda r{:}\left[\mathrm{x}{:}\mathit{Rec}\right] . \left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c} & : & p(\mathrm{x}) \\ \mathrm{e} & : & q(r.\mathrm{x}) \end{array}\right]$

We could compute a fixed point type for this given in (78) but, assuming that individuals are not records, this type will be necessarily empty.

(78) $\left[\begin{array}{lll} \mathrm{x} & : & (\mathit{Rec}\wedge\mathit{Ind}) \\ \mathrm{c} & : & p(\mathrm{x}) \\ \mathrm{e} & : & q(\mathrm{x}) \end{array}\right]$

The problem here is that we have the label ‘x’ occurring both in the domain type and in the returned type and in the fixed point type it is required to label the same object. In general, if we want to construct fixed point types from functions we should avoid cases where the paths in the domain type overlap those in the returned type.

Even if the domain type is a record type and the returned type is a record type and there is no label clash between the two it will not always be possible to construct a fixed point type. Consider the two functions in (79).

(79) a. $\lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right] . \left[\begin{array}{lll} \mathrm{e} & : & p(r.\mathrm{x}) \end{array}\right]$

b. $\lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right] . \left[\begin{array}{lll} \mathrm{e} & : & p(r) \end{array}\right]$

In the case of (79a) we can construct the fixed point type in (80).

(80) $\left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & p(\mathrm{x}) \end{array}\right]$

Suppose that (81) holds.

(81) $r = \left[\begin{array}{lll} \mathrm{x} & = & a \\ \mathrm{e} & = & s \end{array}\right]$

where a : *Ind*
$s : p(a)$

Suppose that $\mathcal{T}$ is (79a). Then $r : \mathcal{T}(r)$, that is, $r : \left[\text{e}{:}p(a)\right]$. Now consider (79b). A candidate for a fixed point would be (82).

(82) $r' = \begin{bmatrix} \text{x} & = & a \\ \text{e} & = & s \end{bmatrix}$

where a : *Ind*
$s : p(r')$

However, the conditions expressed in (82) do not correspond to any type as illustrated in (83).

(83) $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e} & : & p(?) \end{bmatrix}$

In TTR we do not allow for types that are reentrant or non-well-founded where the '?' in (83) could be replaced by something which refers to the whole record which is of the type in (83). This means that there is no way of characterizing a fixed point type for (79b) in TTR. The difference between (79a) and (79b) is that (79a) depends only on paths in r but not on r as a whole, whereas (79b) depends on r as an object. We say that (79a) is *path-dependent* on r whereas(79b) is not. Path dependence is characterized in (84), repeated in the Appendix on p. 498.

(84) A dependent type, $\mathcal{T}$, is *path-dependent on* r, $\mathcal{T}(\!(r)\!)_{\text{path}}$, just in case $\mathcal{T}$ depends on paths in r but not on the whole object r.

For cases like (79b) where path dependence does not obtain there is a related notion of *quasi-fixed point*. Consider the record in (85a) which is of the type (85b).

(85) a. $r = \begin{bmatrix} \mathfrak{c}^* & = & \begin{bmatrix} \text{x} & = & a \end{bmatrix} \\ \text{e} & = & s \end{bmatrix}$

where: a : *Ind*
$s : p(\left[\text{x}{=}a\right])$

b. $\begin{bmatrix} \mathfrak{c}^* & : & \begin{bmatrix} \text{x} & : & \textit{Ind} \end{bmatrix} \\ \text{e} & : & p(\mathfrak{c}^*) \end{bmatrix}$

Supposing that $\mathcal{T}$ is (79b), we now have $r : \mathcal{T}(r.\mathfrak{c}^*)$, that is, $r : \big[\text{e}{=}p(\big[\text{x}{=}a\big])\big]$. We say that r is a quasi-fixed point for $\mathcal{T}$ and that (85b) is a quasi-fixed point type for $\mathcal{T}$ as characterized in (86), repeated in the Appendix on p. 498.

(86) *T is a quasi-fixed point type for dependent type $\mathcal{T}$* iff $r : T$ implies that there is a component $a\varepsilon r$ such that $r : \mathcal{T}(a)$

If we use the distinguished label '$\mathfrak{c}^*$' only used in the construction of quasi-fixed point types, this will also guard against unwanted label clash.

If T is a record type and r is a record of any type, we will use $T_{r.\pi\rightsquigarrow\pi}$ to designate the type like T except that for any $\pi \in \mathrm{paths}(T)$ any occurrence of $r.\pi$ is replaced by π. For example, suppose $T(\!(r)\!)$ is (87) (using abbreviatory notation for dependent fields).

(87) $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog}(r.\text{x}) \end{bmatrix}$

Then $T(\!(r)\!)_{r.\pi\rightsquigarrow\pi}$ will be (88).

(88) $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog}(\text{x}) \end{bmatrix}$

(88) is now no longer dependent on r. Note that using the abbreviatory notation here conveniently hides some additional processing that is going on in this conversion, namely, that a non-dependent field is being converted to a dependent field. In official notation (88) is represented as (89).

(89) $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e} & : & \langle \lambda v{:}\textit{Ind}\ .\ \text{dog}(v), \langle\text{x}\rangle\rangle \end{bmatrix}$

We then characterize $\mathcal{F}$ as in (90), repeated in the Appendix on p. 498.

(90) If $\mathcal{T}$ is a dependent record type of the form $\lambda r{:}T_1\ .\ T_2(\!(r)\!)_{\text{path}}$ where T_1 is a record type and for any r, $\mathrm{paths}(T_1) \cap \mathrm{paths}(\mathcal{T}(r)) = \emptyset$, then

$$\mathcal{F}(\mathcal{T}) \text{ is that type } T \text{ such that for any } r^* : T_1,\ \lambda r{:}T_1\ .\ (T_1 \dot{\wedge} T_2)_{r.\pi\rightsquigarrow\pi}(r^*) = T$$

We illustrate this with an example. Suppose that $\lambda r : T_1 \,.\, T_2(\!(r)\!)_{\text{path}}$ is 'dog′', that is, (91).

(91) $\lambda r{:}\begin{bmatrix}\text{x}:\textit{Ind}\end{bmatrix} \,.\, \begin{bmatrix}\text{e} & : & \text{dog}(r.\text{x})\end{bmatrix}$

Then $\lambda r : T_1 \,.\, T_1 \dot{\wedge} T_2(\!(r)\!)$ is (92).

(92) $\lambda r{:}\begin{bmatrix}\text{x}:\textit{Ind}\end{bmatrix} \,.\, \begin{bmatrix}\text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog}(r.\text{x})\end{bmatrix}$

Removing any dependence on r we obtain $\lambda r : T_1 \,.\, (T_1 \dot{\wedge} T_2)_{r.\pi \rightsquigarrow \pi}$, that is, (93).

(93) $\lambda r{:}\begin{bmatrix}\text{x}:\textit{Ind}\end{bmatrix} \,.\, \begin{bmatrix}\text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog}(\text{x})\end{bmatrix}$

Note that since we have removed all dependencies on r, (93) is a constant function, that is, for any record of type $\begin{bmatrix}\text{x}:\textit{Ind}\end{bmatrix}$ it will return the type (94).

(94) $\begin{bmatrix}\text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog}(\text{x})\end{bmatrix}$

(94) is, then, $\mathcal{F}$(dog′).

The importance of the constraint that there are no shared paths in the domain type and the resulting type of the function to which we apply $\mathcal{F}$ is made clear by the abstract example in (95). Suppose that we did not have this restriction and allowed $\mathcal{F}$ to apply to a function like (95a). Then the putative result would be (95b) which is not well-formed since we do not allow dependent fields which depend on themselves.

(95) a. $\lambda r{:}\begin{bmatrix}\ell{:}T_1\end{bmatrix} \,.\, \begin{bmatrix}\ell & : & T_2(\!(r.\ell)\!)\end{bmatrix}$

b. $\begin{bmatrix}\ell & : & T_1 \dot{\wedge} T_2(\!(\ell)\!)\end{bmatrix}$

If we feel that we need to construct a fixed point type for a dependent type where there are shared paths, then we can achieve the effect we need by first relabelling the dependent type so that there are no shared paths.

We do not need to be careful to prevent clashing of paths in constructing a quasi-fixed point type. Here we replace any path $r.\pi$ with $\mathfrak{c}^*.\pi$. We define the operation constructing a quasi-fixed point type, $\mathcal{F}_{\text{quasi}}$ in (96) where we make use of an additional distinguished label '$\mathfrak{r}^*$'. This is repeated in the Appendix on p. 498.

(96) If $\mathcal{T}$ is a dependent record type of the form $\lambda r\!:\!T_1 \,.\, T_2(\!(r)\!)$ where T_1 is a record type, then

$$\mathcal{F}_{\text{quasi}}(\mathcal{T}) \text{ is that type } T \text{ such that for any } r^* : T_1,$$
$$\lambda r\!:\!T_1 \,.\, (\big[\mathfrak{c}^*\!:\!T_1\big] \wedge\!\!\!\!. \big[\mathfrak{r}^*\!:\!T_2\big])_{r.\pi \rightsquigarrow \mathfrak{c}^*.\pi}(r^*) = T$$

For dependent types which require more than one argument we can define a recursive version of $\mathcal{F}_{\text{quasi}}$, $\mathcal{F}_{\text{quasi}^*}$, in (97), repeated in the Appendix on p. 498.

(97) If $\mathcal{T}$ is a dependent record type of the form $\lambda r\!:\!T_1 \,.\, T_2(\!(r)\!)$ where T_1 is a record type and for any r in its domain $\mathcal{T}(r)$ is a record type, then

$$\mathcal{F}_{\text{quasi}^*}(\mathcal{T}) = \mathcal{F}_{\text{quasi}}((T))$$

else if $\mathcal{T}$ is a dependent record type of the form $\lambda r\!:\!T_1 \,.\, \mathcal{T}'(\!(r)\!)$ where T_1 is a record type and for any r in its domain $\mathcal{T}(r)$ is a dependent record type, then

$$\mathcal{F}_{\text{quasi}^*}(\mathcal{T}) \text{ is that type } T \text{ such that for any } r^* : T_1,$$
$$T = \lambda r\!:\!T_1 \,.\, (\big[\mathfrak{c}^*\!:\!T_1\big] \wedge\!\!\!\!. \big[\mathfrak{r}^*\!:\!\mathcal{F}_{\text{quasi}^*}(\mathcal{T}')\big])_{r.\pi \rightsquigarrow \mathfrak{c}^*.\pi}$$

We shall use the fixed point type of the first argument to restrict the dependent type which is the second argument. We define the restriction of a function by a type as in (98).

(98) If f is a function $\lambda v\!:\!T_1 \,.\, \phi$, then the *restriction of f by a type* T_2, $f|_{T_2}$, is $\lambda v\!:\!(T_1 \wedge\!\!\!\!. T_2) \,.\, \phi$

We can extend this notation to properties as in (99).

(99) If P:*Ppty*, then $P|_T$ is the property

$$\begin{bmatrix} \text{bg} & = & P.\text{bg} \wedge\!\!\!\!. T \\ \text{fg} & = & P.\text{fg}|_T \end{bmatrix}$$

that is,

$\ulcorner P.\text{fg}|_T \urcorner$

We can then define dynamic versions of the contents of quantifier determiners as in (100).

(100) a. SemIndefArt —

$$\lambda Q{:}\mathit{Ppty}\ .\ \ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q & : & \mathit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{bmatrix}\urcorner$$

b. SemDefArt —

$$\lambda Q{:}\mathit{Ppty}\ .\ \ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix}\text{e:unique}(Q)\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q\restriction c.\mathfrak{c}.\text{e} & : & \mathit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty} \\ \text{e} & : & \text{every(restr, scope)} \end{bmatrix}\urcorner$$

The original motivation for treating generalized quantifiers dynamically was to be able to treat the kind of “donkey-anaphora” binding that occurs in sentences like *every farmer who owns a donkey likes it*. Our version of dynamic generalized quantifiers essentially replicates the treatment in Chierchia (1995), though in our own terms. A similar analysis of generalized quantifiers, exploiting contexts in type theory, is given in Fernando (2001). In order to see how our strategy here will facilitate the treatment of donkey anaphora we will have to wait until we have a treatment of anaphora in Chapter 7. The basic strategy is to exploit the conservativity of generalized quantifiers and treat *every farmer who owns a donkey likes it* as *every farmer who owns a donkey is a farmer who owns a donkey and likes it*. This is achieved by restricting the second argument of the quantifier predicate in the manner indicated in (100).

For present purposes the advantage of dynamicizing the generalized quantifiers is that if the first argument property is restricted to be a property of ambient temperature then that restriction will be passed on to the second argument. Let us look in detail at how this will happen. Consider the type in (101).

(101) $\text{every}(\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{AmbTempFrame}\end{bmatrix}\ .\ \begin{bmatrix}\text{e}{\sqsubseteq}s\text{:temperature}(r.\text{x})\end{bmatrix}\urcorner,$
$\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Rec}\end{bmatrix}\ .\ \begin{bmatrix}\text{e:rise}(r.\text{x})\end{bmatrix}\urcorner)$

The result of applying $\mathcal{F}$ to the foreground of the first argument of (101) in order to obtain a fixed point type is given in (102).

$$\text{(102)}\quad \begin{bmatrix} \text{x} & : & \textit{AmbTempFrame} \\ \text{e}\underline{\varepsilon}s & : & \text{temperature(x)} \end{bmatrix}$$

The condition on ‘every’ in (100**??**) requires that we compare the first argument to ‘every’ with the result of restricting the second argument with (102). The foreground of this is given in (103a), which is identical with (103b) (by the definition of restriction) and (103c) (by the definition of merge) and to (103d) (by the definition of merge[6] because *AmbTempFrame* is a subtype of *Rec*).

$$\text{(103) a.}\quad \lambda r{:}\big[\text{x:}\textit{Rec}\big]\ .\ \big[\text{e:rise}(r.\text{x})\big]|_{\begin{bmatrix}\text{x:}\textit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}}$$

$$\text{b.}\quad \lambda r{:}\big[\text{x:}\textit{Rec}\big]\dot{\wedge}\begin{bmatrix}\text{x:}\textit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}\ .\ \big[\text{e:rise}(r.\text{x})\big]$$

$$\text{c.}\quad \lambda r{:}\begin{bmatrix}\text{x:}\textit{Rec}\dot{\wedge}\textit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}\ .\ \big[\text{e:rise}(r.\text{x})\big]$$

$$\text{d.}\quad \lambda r{:}\begin{bmatrix}\text{x:}\textit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}\ .\ \big[\text{e:rise}(r.\text{x})\big]$$

Thus intuitively by choosing to restrict the first argument property to ambient temperature frames we are also restricting the second argument property to ambient temperature frames.

This technique for dynamic quantifiers also has an important consequence if we try to combine frame level and individual level properties. Suppose for example that we are trying to compute the witness condition for *the temperature runs* where *runs* corresponds to the content given in (42a). Then we will have (104) as the foreground of the second argument property.

$$\text{(104) a.}\quad \lambda r{:}\big[\text{x:}\textit{Ind}\big]\ .\ \big[\text{e:run}(r.\text{x})\big]|_{\begin{bmatrix}\text{x:}\textit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}}$$

$$\text{b.}\quad \lambda r{:}\big[\text{x:}\textit{Ind}\big]\dot{\wedge}\begin{bmatrix}\text{x:}\textit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}\ .\ \big[\text{e:run}(r.\text{x})\big]$$

$$\text{c.}\quad \lambda r{:}\begin{bmatrix}\text{x:}\textit{Ind}\dot{\wedge}\textit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}\ .\ \big[\text{e:run}(r.\text{x})\big]$$

[6]For this step we need to take the version of merge in Appendix 12 which contains the two additional clauses taking account of subtypes.

$$\text{d. } \lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\wedge\mathit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}.\ \begin{bmatrix}\text{e:run}(r.\text{x})\end{bmatrix}$$

Here since neither *Ind* nor *AmbTempFrame* are a subtype of the other the final step of merging represented in (104d) is the meet type (without the dot!) whose components are the two types which were merged. The property represented in (104) is thus necessarily empty (that is, its property extension is the empty set no matter what we assign to the basic types), if we have the assumption that individuals are non-records. This would be a way of requiring that the content of *runs* be coerced to something which could hold for temperature frames in order to prevent the sentence from being anomalous. Similarly, if we wish to find a content for *the dog rises* then we have to associate *rises* with an individual property or alternatively associate *dog* with a frame property.

What then should be the content of *is ninety*? An obvious modification to the treatment of *be* in Chapter 3, substituting the type *Real* for the type *Ind*, would lead to the property in (105).

$$\text{(105) } \ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Real}\end{bmatrix}.\ \begin{bmatrix}\text{x}{=}r.\text{x, 90} & : & \mathit{Ind}\\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner$$

This property might be the content you need if you are treating a sentence like *2 times 45 is 90*. However, if we use this content with *the temperature* we will run into a similar problem as that represented in (104). This is spelled out in (106)

$$\text{(106) a. } \lambda r{:}\begin{bmatrix}\text{x:}\mathit{Real}\end{bmatrix}.\ \begin{bmatrix}\text{x}{=}r.\text{x, 90:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}\Big|\begin{bmatrix}\text{x:}\mathit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}$$

$$\text{b. } \lambda r{:}\begin{bmatrix}\text{x:}\mathit{Real}\end{bmatrix}\dot{\wedge}\begin{bmatrix}\text{x:}\mathit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}.\ \begin{bmatrix}\text{x}{=}r.\text{x, 90:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}$$

$$\text{c. } \lambda r{:}\begin{bmatrix}\text{x:}\mathit{Real}\dot{\wedge}\mathit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}.\ \begin{bmatrix}\text{x}{=}r.\text{x, 90:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}$$

$$\text{d. } \lambda r{:}\begin{bmatrix}\text{x:}\mathit{Real}\wedge\mathit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}.\ \begin{bmatrix}\text{x}{=}r.\text{x, 90:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}$$

Assuming that real numbers are not records, we have the same problem as we had in (104) in that the property turns out to be necessarily empty. What we need instead is a property of frames (records) that will make reference to a scale, ζ, of the kind we defined for *AmbTempFrame* in (16), for example, a property with the foreground given in (107).

(107) $\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Rec}\end{bmatrix} . \begin{bmatrix}\text{x=}\zeta(r.\text{x}),\ 90{:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}$

If ζ is fixed to be the scale in (16) then (107) is identical with (108).

(108) $\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Rec}\end{bmatrix} . \begin{bmatrix}\text{x=}r.\text{x.x},\ 90{:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}$

That is, what is checked for being identical with 90 is the 'x'-field of the temperature frame which is in the 'x'-field of the argument to the property. If we choose this property as the content for *is ninety* then the restriction of the property as second argument to the quantifier will give a property as result which is not necessarily empty. This property is shown in (109).

(109) a. $\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Rec}\end{bmatrix} . \begin{bmatrix}\text{x=}\zeta(r.\text{x}),\ 90{:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}\Big|\begin{bmatrix}\text{x:}\mathit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix}\urcorner$

b. $\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Rec}\end{bmatrix}\dot{\wedge}\begin{bmatrix}\text{x:}\mathit{AmbTempFrame}\\ \text{e=}s\text{:temperature(x)}\end{bmatrix} . \begin{bmatrix}\text{x=}\zeta(r.\text{x}),\ 90{:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}\urcorner$

c. $\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Rec}\dot{\wedge}\mathit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix} . \begin{bmatrix}\text{x=}\zeta(r.\text{x}),\ 90{:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}\urcorner$

d. $\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{AmbTempFrame}\\ \text{e}\underline{\varepsilon}s\text{:temperature(x)}\end{bmatrix} . \begin{bmatrix}\text{x=}\zeta(r.\text{x}),\ 90{:}\mathit{Ind}\\ \text{e:be(x)}\end{bmatrix}\urcorner$

Now, as in (103), (109d) is equivalent to (109c) in the sense that exactly the same objects will have the properties. This is because *AmbTempFrame* is a subtype of *Rec*. In the functions in (109) there are two parameters which will need to be determined by context in compositional semantics, that is, will need to be found by matching the domain type of a parametric content against an agent's resources. These are the resource situation, s, and the scale, ζ.

5.6 Individual vs. frame level nouns

We have made a distinction between individual level nouns like *dog* and frame level nouns like *temperature*, differentiating their contents as in (42) and motivating the distinction with the Partee puzzle. Now consider (110).

(110) a. The dog is nine

b. The dog is getting older/aging

c. Nine is getting older/aging

We have the same intuitions about (110) as we do about the original temperature puzzle. We cannot conclude (110c) from (110a,b). Does this mean that *dog* is a frame level noun after all? Certainly, if we think of frames as being like entries in relational databases, it would be natural to think of age (or information allowing us to compute age such as date of birth) as being a natural field in a dog-frame.[7]

Our strategy to deal with this will be to say that contents of individual level nouns can be coerced to frame level contents, whereas the contents of frame level nouns cannot be coerced "down" to individual level contents. Thus in addition to (111a) we have (111b).

(111) a. $\ulcorner\lambda c{:}\mathit{Cntxt}\,.\,\ulcorner\lambda r{:}\big[\mathrm{x}{:}\mathit{Ind}\big]\,.\,\big[\ \mathrm{e}\ :\ \mathrm{dog}(r.\mathrm{x})\ \big]\urcorner\urcorner$

b. $\ulcorner\lambda c{:}\mathit{Cntxt}\,.\,\ulcorner\lambda r{:}\big[\mathrm{x}{:}\mathit{Rec}\big]\,.\,\big[\ \mathrm{e}\ :\ \mathrm{dog\_frame}(r.\mathrm{x})\ \big]\urcorner\urcorner$

The predicate 'dog_frame' is related to the predicate 'dog' by the constraint in (112).

(112) e : dog_frame(r) implies r : $\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\\ \mathrm{e}{:}\mathrm{dog}(\mathrm{x})\end{bmatrix}$

There are several different kinds of dog frames with additional information about a dog which an agent may acquire or focus on. Here we will consider just frames which contain a field labelled 'age' as exemplified in (113).

(113) $\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\\ \mathrm{e}{:}\mathrm{dog}(\mathrm{x})\\ \mathrm{age}{:}\mathit{Real}\\ \mathrm{c_{age}}{:}\mathrm{age\_of}(\mathrm{x},\mathrm{age})\end{bmatrix}$

An age scale, ζ_{age}, for individuals can then be defined as the function in (114).

(114) $\zeta_{\mathrm{age}} = \lambda r{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\\ \mathrm{age}{:}\mathit{Real}\\ \mathrm{c_{age}}{:}\mathrm{age\_of}(\mathrm{x},\mathrm{age})\end{bmatrix}\,.\ r.\mathrm{age}$

[7] Curiously, it does not seem to figure in FrameNet for *dog* (as of 2nd March, 2015). The noun *dog* is associated with the frame Animals which inherits from the frame Biological entity. But in neither of these frames is there a frame element corresponding to age or date of birth. There is a frame Age but this does not seem to be related to Animals or Biological entity.

The content for *is nine* in *the dog is nine* is then like (107) with ζ set to ζ_{age} and 9 replacing 90. Thus *be* followed by a numeral can be coerced to a content depending on some scale which is available as a resource.

We can think of the sentence *the dog is nine* as involving two coercions: one coercing the content of *dog* to a frame level property and the other coercing the content of *be* to a function which when applied to a number will return a frame level property depending on an available scale. Such coercions do not appear to be universally available in languages. For example, in German it is preferable to say *die Temperatur ist 35 Grad* "the temperature is 35 degrees" rather than *#die Temperatur ist 35* "the temperature is 35". Similarly *der Hund ist neun Jahre alt* "the dog is nine years old" is preferred over *#der Hund ist neun* "the dog is nine". We will return to the matter of coercion or creation of new contents in Section 5.7.

We can think of the common noun sign types like (111a) as unmodulated in something like the sense of modulation discussed by Recanati (2010) in that the restriction type yielding the type of the domain of the property is identical with the type that represents the arity of the predicate. We will see later a way to modulate the content of the noun by choosing a subtype of the type of the predicate argument as the domain type of the property.

Here we discuss an operation 'CommonNounIndToFrame' which is defined on individual level common noun sign types and "raises" them to frame level common noun sign types. In order to facilitate this we first introduce a function $\mathrm{FrameType}$ which maps predicates with arity $\langle \mathit{Ind} \rangle$ to a type of frames. The way $\mathrm{FrameType}$ is defined depends on a particular agent at a particular time, that is, $\mathrm{FrameType}(p)$ represents the type of frames that an agent associates with the predicate p. A general constraint on $\mathrm{FrameType}$ is that for any p in its domain (115) must hold.

(115) $\mathrm{FrameType}(p) \sqsubseteq \begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & p(\mathrm{x}) \end{bmatrix}$

Using this we can derive a new predicate 'p_frame' from any predicate p for which $\mathrm{FrameType}$ is defined. This is characterized in (116).

(116) a. If p is a predicate in the domain of $\mathrm{FrameType}$, then p_frame is a predicate with arity $\langle \mathit{Rec} \rangle$.

b. $e : p\text{\_frame}(r)$ iff $r : \mathrm{FrameType}(p)$ and $e = r$

'CommonNounIndToFrame' is defined in (117).

(117) If T_{phon} is a phonological type, p is a predicate with arity $\langle \mathit{Ind} \rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then

CommonNounIndToFrame(Lex$_{\text{CommonNoun}}$(T_{phon}, T_{bg}, p)) =

Lex$_{\text{CommonNoun}}$(T_{phon}, T_{bg}, p_frame)

This operation is a universal resource which may or may not be used by individual languages. Given the discussion in Section 5.6, we suggest that it is used productively in English but not in German, for example. This gives us a way of generating new lexical resources from already existing resources. Similarly, we can think of p_frame as being the result of applying a "raising" operation to the predicate p.

Another way to generate new lexical resources from basic common noun sign types is to restrict the domain of the common noun by some type (perhaps related to a topos as suggested in Section 5.4). This is formulated in (118).

(118) If T_{phon} is a phonological type, p is a predicate, T_{bg} and T_{res} are record types and Σ is Lex$_{\text{CommonNoun}}$(T_{phon}, T_{bg}, p), then RestrictCommonNoun(Σ, T_{res}) is

$$\Sigma \boxed{\dot{\wedge}} \left[\ \text{cont}{=}\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\text{SemCommonNoun}(T_{\text{bg}}, p)(c)\,|_{T_{\text{res}}}\urcorner\urcorner \quad : \quad \textit{PPpty}\ \right]$$

This will enable us, for example, to restrict the basic lexical entry for *temperature*, $\Sigma_{\text{“temperature”}}$ (repeated in (119a)) to obtain the additional lexical resource (119b).

(119) a. $\Sigma_{\text{“temperature”}}$ = Lex$_{\text{CommonNoun}}$("temperature", *Rec*, temperature)

b. RestrictCommonNoun($\Sigma_{\text{“temperature”}}$, *AmbTempFrame*)

This gives us a content for *temperature* which is restricted to ambient temperature.

We can combine restriction coercion with frame coercion. While frame coercion gives us a general frame level property of records we can restrict the frame to be of a certain type corresponding to a particular type of frame that we have as a resource. For example, suppose that we have resource which is a frame type for dog frames, *DogFrame* as introduced in (113) repeated in (120).

(120) $\begin{bmatrix} \text{x}:\textit{Ind} \\ \text{e}:\text{dog(x)} \\ \text{age}:\textit{Real} \\ \text{c}_{\text{age}}:\text{age\_of(x,age)} \end{bmatrix}$

We can use *DogFrame* to restrict the result of coercing our frame level dog sign type, that is there can be a two-step coercion from the basic lexical entry in (121a), $\Sigma_{\text{“dog”}}$, as represented in (121a) to (121b).

(121) a. $\Sigma_{\text{“dog”}} = \text{Lex}_{\text{CommonNoun}}(\text{“dog”}, \mathit{Rec}, \text{dog})$

b. RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“dog”}}$), *DogFrame*)

We need a treatment of *is* which will allow it to combine with numerals like *nine* and *ninety* to form a frame level predicate as indicated in (107). We start from a parametrized version of the definition of SemBe which we introduced in Chapter 3, also adjusted for our new treatment of properties. This is given in (122).

(122)
$$\ulcorner\lambda c : T_{\text{bg}} \;.\; \lambda \mathcal{Q}{:}\mathit{Quant} \;.\; \ulcorner\lambda r_1{:}\begin{bmatrix}\text{x}{:}\mathit{Ind}\end{bmatrix} \;.\; \mathcal{Q}(\ulcorner\lambda r_2{:}\begin{bmatrix}\text{x}{:}\mathit{Ind}\end{bmatrix} \;.\; \begin{bmatrix}\text{x}{=}r_1.\text{x}, r_2.\text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner)\urcorner\urcorner$$

Here the context represented by the first argument to the function, c, does not contribute anything to the final content of *be* which represents straightforward equality. In this chapter we want to allow equality not only between individuals but also objects of other types. We can do this by letting the content of *be* be parametric on a type including the label ‘x’ introduced in the context as in (123).

(123)
$$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix}\text{ty}{:}\mathit{Type}\end{bmatrix}\end{bmatrix} \;.\; \lambda \mathcal{Q}{:}\mathit{Quant} \;.\; \ulcorner\lambda r_1{:}\begin{bmatrix}\text{x}{:}c.\mathfrak{c}.\text{ty}\end{bmatrix} \;.\; \mathcal{Q}(\ulcorner\lambda r_2{:}\begin{bmatrix}\text{x}{:}c.\mathfrak{c}.\text{ty}\end{bmatrix} \;.\; \begin{bmatrix}\text{x}{=}r_1.\text{x}, r_2.\text{x} & : & c.\mathfrak{c}.\text{ty} \\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner)\urcorner\urcorner$$

This gives us a content for *be* which will express identity for objects in any type. Thus it will be appropriate for both sentences in (124) where (124a) expresses identity for individuals and (124b) expresses identity for numbers.

(124) a. The dog is Fido

b. The number is nine

We will call this ‘SemBe_{ID}’. We need a slightly different content in order to deal with (125).

(125) a. The dog is nine

b. The temperature is ninety

Here we need to introduce a scale into the context and make sure that the second argument to *be* is a number. This is given in (126).

$$(126)\ \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{ty}{:}\textit{Type}\\ \text{sc}{:}(\text{ty}\to\textit{Real})\end{bmatrix}\end{bmatrix}\ .\\ \lambda\mathcal{Q}{:}\textit{Quant}\ .\\ \ulcorner\lambda r_1{:}\begin{bmatrix}\text{x}{:}c.\mathfrak{c}.\text{ty}\end{bmatrix}\ .\\ \mathcal{Q}(\ulcorner\lambda r_2{:}\begin{bmatrix}\text{x}{:}\textit{Real}\end{bmatrix}\ .\ \begin{bmatrix}\text{x}{=}c.\mathfrak{c}.\text{sc}(r_1.\text{x}),\ r_2.\text{x} & : & \textit{Real}\\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner)\urcorner\urcorner$$

We will refer to this as ‘$\text{SemBe}_{\text{scalar}}$’.

We now have two similar contents for *be*. It would be a simple matter to define ‘$\text{SemBe}_{\text{scalar}}$’ as the result of a coercion operation applied to ‘SemBe_{ID}’ if we wanted to. Montague (1973) treated both kinds of *be* as identical and used individual concepts instead of individuals in order to account for the Partee puzzle. Most following work avoids this use of individual concepts and instead assumes that the content of *be* is different in the two cases. The analysis we have presented is a way of capturing both intuitions. We have two contents associated with *be* but we have made explicit that they are closely related and both basically have *be* express an identity relation, corresponding to Montague’s intuition. Note that technically both sentences in (127) are ambiguous according to this analysis.

(127) a. The dog is nine

b. The temperature is ninety

They each have a nonsense necessarily false reading in which, respectively, the dog is identical with the number nine and the temperature is identical with the number ninety. The derivation of both of these sentences involves a property application which results in ‘$\perp$’ (see (50) on p. 232). It is natural to assume that speakers in general disprefer the content ‘$\perp$’. Note that this does not necessarily mean that speakers disprefer all contradictory readings. There are other necessarily empty types. For example, suppose that T_1 and T_2 preclude each other, that is, no object can be a witness for both types. Then $T_1 \wedge T_2$ is a necessarily empty type but is distinct from ‘$\perp$’.

To complete the picture we need to account for *nine* and *ninety*. We will treat these as logically proper names of real numbers. Thus we will not treat them as introducing presuppositions in the manner in which we suggested in Chapter 4 but rather in the Montague-like manner which we used for proper names in Chapter 3, except that we now adjust it to take account of parametric contents and the new definition of *Ppty* (property). In n is a (real) number, then SemNumeral(n) (the content for a number expression such as *nine*) is as given in (128).

(128) $\ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda P{:}\mathit{Ppty}\ .\ P(\begin{bmatrix}\mathrm{x}{=}n\end{bmatrix})\urcorner$

Then we can define $\mathrm{Lex}_{\mathrm{numeral}}$ as an operation which takes a phonological type T_{phon} and a (real) number n and returns the sign type (129).

(129) $\mathrm{Lex}_{\mathrm{numeral}}(T_{\mathrm{phon}}, n) =$
$\quad\mathrm{Lex}(T_{\mathrm{phon}}, \mathit{NP}) \wedge\!\!\!\!.\ \begin{bmatrix}\mathrm{cnt}{=}\mathrm{SemNumeral}(n){:}\mathit{PQuant}\end{bmatrix}$

The two sign types that we need as resources for our small fragment are given in (130).

(130) a. $\mathrm{Lex}_{\mathrm{numeral}}$(“nine”, 9)
b. $\mathrm{Lex}_{\mathrm{numeral}}$(“ninety”, 90)

5.7 Passengers and ships

Gupta (1980) points out examples such as (131).

(131) a. National Airlines served at least two million passengers in 1975
b. Every passenger is a person
c. National Airlines served at least two million persons in 1975

His claim is that we cannot conclude (131c) from (131a,b). There is a reading of (131a) where what is being counted is not passengers as individual people but passenger events, events of people taking flights, where possibly the same people are involved in several flights. Gupta claims that it is the only reading that this sentence has. While it is certainly the preferred reading for this sentence (say, in the context of National Airlines' annual report or advertizing campaign), I think the sentence also has a reading where individuals are being counted. Consider (132).

(132) National Airlines served at least two million passengers in 1975. Each one of them signed the petition.

While (132) could mean that a number of passengers signed the petition several times our knowledge that people normally only sign a given petition once makes a reading where there are two million distinct individuals involved more likely. Similarly, while (131c) seems to prefer the individual reading where there are two million distinct individuals it is not impossible to get an event reading here. Krifka (1990) makes a similar point. Gupta's analysis of such examples involves individual concepts and is therefore reminiscent of the functional concepts used by Löbner (1979, 1981) to analyze the Partee puzzle.

Carlson (1982) makes a similar point about Gupta's examples in that nouns which appear to normally point to individual related readings can in the right context get the event related readings. One of his examples is a traffic engineer's report as in (133).

(133) Of the 1,000 cars using Elm St. over the past 49 hours, only 12 cars made noise in excess of EPA recommended limits.

It is easy to interpret this in terms of 1,000 and 12 car events rather than individual cars. Carlson's suggestion is to use his notion of *individual stage*, what he describes intuitively as "things-at-a-time". Krifka (1990) remarks that "Carlson's notion of a stage serves basically to reconstruct events". While this is not literally correct, the intuition is nevertheless right. Carlson was writing at a time when times and time intervals were used to attempt to capture phenomena that in more modern semantics would be analyzed in terms of events or situations. Thus Carlson's notion of stage is related to a frame-theoretic approach which associates an individual with an event.

Consider the noun *passenger*. It would be natural to assume that passengers are associated with journey events. FrameNet[8] does not have an entry for *passenger*. The closest relevant frame appears to be TRAVEL which has frame elements for traveller, source, goal, path, direction, mode of transport, among others. The FrameNet lexical entry for *journey* is associated with this frame. Let us take the type *TravelFrame* to be the stripped down version of the travel frame type in (134a). Then we could take the type *PassengerFrame* to be (134b).

(134) a. $\begin{bmatrix} \text{traveller} & : & \mathit{Ind} \\ \text{source} & : & \mathit{Loc} \\ \text{goal} & : & \mathit{Loc} \end{bmatrix}$

b. $\begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{passenger(x)} \\ \text{journey} & : & \mathit{TravelFrame} \\ \text{c}_{\text{travel}} & : & \text{take\_journey(x, journey)} \end{bmatrix}$

[8]As of 13th May 2015.

Here 'take_journey' is a predicate with arity $\langle \mathit{Ind}, \mathit{TravelFrame} \rangle$ with the witness condition in (135).

(135) If a:*Ind* and e:*TravelFrame*, then

$$s : \text{take\_journey}(a, e) \text{ iff } s = e \text{ and } e.\text{traveller} = a$$

Let us suppose that the basic lexical entry for *passenger* is (136), where 'passenger' is a predicate with arity $\langle \mathit{Ind} \rangle$.

(136) $\text{Lex}_{\text{CommonNoun}}$("passenger", *Rec*, passenger)

Let us call (136) $\Sigma_{\text{“passenger”}}$. Just as we did for *dog* we can introduce a coerced version of the lexical entry for *passenger* as in (137), using the type *PassengerFrame*.

(137) RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“passenger”}}$), *PassengerFrame*)

This means that the non-parametric content will now be a property of passenger frames of type *PassengerFrame*. This introduces not only a passenger but also a journey, an event in which in which the passenger is the traveller.

It seems that we have now done something which Krifka (1990) explicitly warned us against. At the end of his discussion of Carlson's analysis he comes to the conclusion that it is wrong to look for an explanation of event-related readings of these sentences in terms of a noun ambiguity. One of Krifka's examples is (138) (which gives the title to his paper).

(138) Four thousand ships passed through the lock

This can either mean that four thousand distinct ships passed through the lock or that the there were four thousand ship-passing-through-the-lock events a number of which involved the same ships. The problem he sees is that if we treat *ship* as being ambiguous between denoting individual ships or ship stages in Carlson's sense then there will be too many stages which pass through the lock. For example, suppose that a particular ship passes through the lock twice. This gives us two stages of the ship which pass through the lock. But then, Krifka claims, there will be a third stage, the sum of the first two, which also passes through the lock. It is not clear to me that this is an insuperable problem for the stage analysis. We need to count stages that pass through the lock exactly once. Let us see how the frame analysis fares.

We will start with a singular example in order to avoid the additional problems offered by the plural. Consider (139).

(139) Every passenger gets a hot meal

Suppose that an airline has this as part of its advertizing campaign. Smith, a frequent traveller, takes a flight with the airline and as expected gets a hot meal. A few weeks later she takes another flight with the same airline and does not get a hot meal. She sues the airline for false advertizing. At the hearing, her lawyer argues, citing Gupta (1980), that the advertizing campaign claims that every passenger gets a hot meal on every flight they take. The lawyer for the airline company argues, citing Krifka (1990), that the sentence in question is ambiguous between an individual and an event reading, that the airline had intended the individual reading and thus the requirements of the advertizing campaign had been met by the meal that Smith was served on the first flight. Smith's lawyer then calls an expert witness, a linguist who quickly crowdsources a survey of native speakers' interpretations of the sentence in the context of the campaign and discovers that there is an overwhelming preference for the meal-on-every-flight reading. (The small percentage of respondents who preferred the individual reading over the event reading gave their occupation as professional logician.) Smith wins the case and receives an additional hot meal.

What is important for us at the moment is the fact that there is an event reading of this sentence. We will return to the matter of preferred readings below. We will treat the content of *every* on the model of the content of the indefinite article, except that the quantifier relation will be 'every' instead of 'exist'. Thus we will define SemUniversal on the model of SemIndefArt.[9] This is given in (140).

(140) $\lambda Q{:}\mathit{Ppty}\ .$
$$\ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \left[\begin{array}{lll} \text{restr}{=}Q & : & \mathit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty} \\ \text{e} & : & \text{every(restr, scope)} \end{array}\right]\urcorner$$

If we use the content associated with *passenger* in (137) the content associated with *every passenger* will be (141).

(141) $\ulcorner\lambda c{:}\mathit{Cntxt}\ .$

[9] We leave to one side the issue of whether *every* should introduce a background constraint that there are at least three objects which have the property associated with the noun.

$$\lambda P{:}\mathit{Ppty}\ .\ \left[\begin{array}{lcl} \text{restr}{=}\ulcorner\lambda r{:}\left[\text{x}{:}\mathit{PassengerFrame}\right]\ .\ \text{passenger\_frame}(r.\text{x})\urcorner & : & \mathit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty} \\ \text{e} & : & \text{every(restr, scope)} \end{array}\right]\urcorner$$

In order to simplify matters let us treat *gets a hot meal* as if it were an intransitive verb corresponding to a single predicate 'get_a_hot_meal'. This is a predicate whose arity is $\langle \mathit{Ind} \rangle$. It is individuals, not frames (situations), that get hot meals. Thus the content of *gets a hot meal* will be (142).

(142) $\ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \ulcorner\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \left[\ \text{e}\ :\ \text{get\_a\_hot\_meal}(r.\text{x})\ \right]\urcorner\urcorner$

While (142) is the right type of argument for (141) since it is a property it will lead us eventually into problems because there is nothing which is both a passenger frame and an individual for the reasons discussed in Section 5.5. What we need is a coercion which will obtain a frame level intransitive verb to match the frame level noun. This would be a coercion IntransVerbIndToFrame exactly parallel to CommonNounIndToFrame defined in (117). Thus IntransVerbIndToFrame is defined as in (143).

(143) If T_{phon} is a phonological type, p is a predicate with arity $\langle \mathit{Ind} \rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then
IntransVerbIndToFrame($\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$$\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p\text{\_frame})$$

Thus the new parametric content derived for *get_a_hot_meal* will be (144).

(144) $\ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \ulcorner\lambda r{:}\left[\text{x}{:}\mathit{Rec}\right]\ .\ \left[\ \text{e}\ :\ \text{get\_a\_hot\_meal\_frame}(r.\text{x})\ \right]\urcorner\urcorner$

A "get_a_hot_meal" frame will, according to our characterization of predicates of the form 'p_frame' be of type (145).

(145) $\left[\begin{array}{lcl} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{get\_a\_hot\_meal(x)} \end{array}\right]$

Intuitively the 'every' relation holding between the two frame-level coerced individual properties corresponding to *passenger* and *get_a_hot_meal* will mean "every frame (situation) containing an individual in the 'x'-field who is a passenger taking a journey will be a frame where the individual in the 'x'-field gets a hot meal". Or, more formally, (146).

(146) every r of type $\begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{passenger(x)} \\ \text{journey} & : & \mathit{TravelFrame} \\ \text{c}_{\text{travel}} & : & \text{take\_journey(x, journey)} \end{bmatrix}$ is of type $\begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{get\_a\_hot\_meal(x)} \end{bmatrix}$

This means that every frame of type *PassengerFrame* will be of type (147a), that is (147b) which is identical with (147c).

(147) a. $\mathit{PassengerFrame} \dot{\wedge} \begin{bmatrix} \text{x:}\mathit{Ind} \\ \text{e:get\_a\_hot\_meal(x)} \end{bmatrix}$

b. $\begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{passenger(x)} \\ \text{journey} & : & \mathit{TravelFrame} \\ \text{c}_{\text{travel}} & : & \text{take\_journey(x, journey)} \end{bmatrix} \dot{\wedge} \begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{get\_a\_hot\_meal(x)} \end{bmatrix}$

c. $\begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{passenger(x)} \wedge \text{get\_a\_hot\_meal(x)} \\ \text{journey} & : & \mathit{TravelFrame} \\ \text{c}_{\text{travel}} & : & \text{take\_journey(x, journey)} \end{bmatrix}$

Thus even though we have coerced to a frame-level reading it is still the passengers (i.e. individuals) in the frames who are getting the hot meal not the situation which is the frame.

Things go less well with cardinality quantifiers, however. Consider *2000 passengers get a hot meal* which corresponds to (148).

(148) 2000 r of type $\begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{passenger(x)} \\ \text{journey} & : & \mathit{TravelFrame} \\ \text{c}_{\text{travel}} & : & \text{take\_journey(x, journey)} \end{bmatrix}$ are of type $\begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{get\_a\_hot\_meal(x)} \end{bmatrix}$

The problem is not exactly the same as the problem which Krifka foresaw with the summing of stages although it is intuitively related. It has to do with the way we have set up subtyping with record types. Given a record of a type we can always add a new field to the record and obtain a distinct record of the same type. Trivially the field we add could contain an object already occurring in a field in the original record. As we are assuming that the set of labels is countably infinite if there is one record of a given type there will be infinitely many records of the same type. We illustrate this with an abstract example in (149).

(149) a. $\begin{bmatrix} \ell_1 & : & T_1 \\ \ell_2 & : & T_2 \end{bmatrix}$

b. $\begin{bmatrix} \ell_1 & = & a \\ \ell_2 & = & b \end{bmatrix}$

c. $\begin{bmatrix} \ell_1 & = & a \\ \ell_2 & = & b \\ \ell_3 & = & a \end{bmatrix}$

d. $\begin{bmatrix} \ell_1 & = & a \\ \ell_2 & = & b \\ \ell_3 & = & a \\ \ell_4 & = & a \end{bmatrix}$

e. ...

If (149b) is of type (149a) (i.e. $a : T_1$ and $b : T_2$), then so are (149c) and (149d) and so on as we successively "grow" the record without changing the fields that make the records a witness for the type and without necessarily adding anything new in the new fields. If records model events (situations) then this corresponds to the intuition that given any event there will always be a larger event of which it is a part. For example, if I wash my hands that is part of an event in which I wash my hands and stand at the washbasin. This is in turn part of an event in which I wash my hands, stand at the washbasin and breathe and so on. We want this to be true but still there is the robust intuition that we are only talking about one event of washing my hands here which is part of infinitely many larger events.

Fortunately, this problem is easy to fix. We use the notion of one record being a proper part of another, that is the set of paths of the first is a proper subset of the set of paths of the second and the objects at the end of the total paths of the first are the same as those at the end of the corresponding total path in the second.

Technically this is given by the definition in (150) (repeated in Appendix 5).

(150) r_1 *is a proper part of* r_2, $r_1 < r_2$, just in case

1. r_1 and r_2 are records,
2. $\text{paths}_{\text{rec}}(r_1) \subset \text{paths}_{\text{rec}}(r_2)$ and
3. for all $\pi \in \text{tpaths}(r_1)$, $r_1.\pi = r_2.\pi$

This notion yields a notion of minimal object of a given type which is related to Schubert's (2000) notion of characterization discussed on p. 236. It is different from Schubert's notion in that we do not say that there are no other types to which the situation belongs but rather that no proper part of the situation is of the type. In this way it is related to the notion of minimal situation discussed by Kratzer (2014) and elsewhere in earlier work. It is also, of course, related to mereological approaches that have been used, for example, in approaches to the analysis of the plural as in Krifka (1990) and much other literature. It is this we will exploit in our analysis of the plural cardinality quantifiers. We introduce the notion of a *plurality* as a set of objects of a given type that does not contain two objects where one is a proper part of the other.

We characterize a notion of plurality types as in (151) (repeated in Appendix 5).

(151) A system of complex types **TYPE**$_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ with set types *has plurality types* if

1. for any $T \in$ **Type**, $\text{plurality}(T) \in$ **Type**
2. for any $T \in$ **Type**, $A :_{\mathbf{TYPE_C}} \text{plurality}(T)$ iff
   a) $A :_{\mathbf{TYPE_C}} \text{set}(T)$
   b) if $a \in A$ then for any b such that $a < b$, $b \notin A$

In terms of our informal proof theoretic notation this can be expressed as (152).

(152) For Γ a system of complex types with set types,

a. $$\frac{\Gamma \vdash T \in \textbf{Type}}{\Gamma \vdash \text{plurality}(T) \in \textbf{Type}}$$

b. $$\frac{\Gamma \vdash A : \text{set}(T) \qquad \begin{array}{c}[a \in A, a < b]\\ \vdots \\ b \notin A\end{array}}{\Gamma \vdash A : \text{plurality}(T)}$$

It might seem natural to require that a plurality contains at least two objects. The choice not to place this requirement on a plurality makes this analysis number neutral in the sense of Zweig (2008, 2009). Zweig (2008) contains a useful overview of some of the variants of analyses of the plural that have been proposed in the literature, including the distinction between set-based and sum-based analyses. In the type theory we have proposed we have sets already available and a kind of mereology based on the structure of records, as illustrated in (150), and we have used a combination of these in our characterization of plurality. Whether this proposal would survive an in-depth investigation of the plural in this framework is an open question. In particular the work on mass terms by Sutton and Filip (2017) that we will in any case need an additional sum-structure.

We propose here a treatment of basic plural quantification cases involving cardinality quantification that will allow us to say something about the content of *2000 passengers get a hot meal*. The reading of the sentence we are treating here is distributive, that is, a reading on which each of the passengers individually gets a hot meal. We will first introduce plural predicates with distributive interpretations. For simplicity, we will define this just for one-place predicates though it could be easily generalized to n-place predicates. If p is a singular predicate then we use 'p_pl' to represent a plural version of p in (153).

(153) a. If p is a singular predicate (i.e. there is no p' such that $p = p'$_pl) with arity $\langle T\rangle$, then p_pl is a predicate with arity $\langle \text{plurality}(T)\rangle$

b. $e : p\text{\_pl}(A)$ if for all $a \in A$, $e : p(a)$

(We use a conditional in (153b) rather than a biconditional because ultimately we think that there will be other non-distributive witness conditions for types constructed with plural predicates.)

We then introduce, in (154), resources 'PluralCommonNoun' and 'PluralIntransVerb' to generate plural nouns and verbs. For this, we will assume without definition that we have operations 'pluralnoun' and 'pluralverb' which will map one (singular) phonological type to another. (We will not in this book present a formulation of morphology in TTR.)

(154) a. If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T\rangle$ and T_{bg} is a record type then PluralCommonNoun($\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$\text{Lex}_{\text{CommonNoun}}(\text{pluralnoun}(T_{\text{phon}}), T_{\text{bg}}, p\text{\_pl})$

b. If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T\rangle$ and T_{bg} is a record type then PluralIntransVerb($\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$\text{Lex}_{\text{IntransVerb}}(\text{pluralverb}(T_{\text{phon}}), T_{\text{bg}}, p\text{\_pl})$

We introduce a type of plural properties, *PlPpty*, considered as a property of pluralities. Its witness condition is given in (155).

(155) P : *PlPpty* iff P : *Ppty* and for some type T, P.bg $\sqsubseteq \left[\text{x:plurality}(T)\right]$

We consider cardinality quantifiers such as *two* and *two thousand* to correspond to predicates whose arity is $\langle$*PlPpty*, *PlPpty*$\rangle$. If n is a natural number (that is, an object of type *Nat*), let 'exactly_n' be such a predicate. Similarly, we introduce predicates 'at_least_n' and 'at_most_n'. In order to give the witness conditions for these predicates we use a predicate 'card' ("cardinality") with arity *Card*$\rangle$, where *Card* is the type of cardinal numbers (the natural numbers together with the transfinite cardinals, $\aleph_0, \aleph_1, \ldots$). A set, X, is of type 'card(n)' if its cardinality is n. Similarly we introduce predicates 'card_at_least' and 'card_at_most' The relevant witness conditions are spelt out in (156).

(156) a. X : card(n) iff for some T, $X : \text{set}(T)$ and $|X| = n$

b. X : card_at_least(n) iff for some T, $X : \text{set}(T)$ and $|X| \geq n$

c. X : card_at_most(n) iff for some T, $X : \text{set}(T)$ and $|X| \leq n$

We now give the witness conditions for the cardinal quantifier predicates in (157).

(157) a. s : exactly_n(P, Q) iff s : at_least_n(P, Q)$\wedge$at_most_n(P, Q)

b. s : at_least_n(P, Q) iff $[\check{}\mathcal{F}((Q\restriction s).\text{fg}\mid_{\mathcal{F}(P.\text{fg})}) \mathbin{\dot{\wedge}} \left[\text{x:card\_at\_least}(n)\right]] \neq \emptyset$

c. s : at_most_n(P, Q) iff $r : \mathcal{F}((Q\restriction s).\text{fg}\mid_{\mathcal{F}(P.\text{fg})})$ implies $r : \left[\text{x:card\_at\_most}(n)\right]$

Let us take these definitions through our example *2000 passengers get a hot meal*. One relevant type which could serve as the content of this sentence is given in (158).

$$\text{(158) at\_least\_2000}(\ulcorner\lambda r{:}\left[\text{x:plurality}(\textit{Rec})\right] .\ \left[\text{e:passenger\_frame\_pl}(r.\text{x})\right]\urcorner,$$
$$\ulcorner\lambda r{:}\begin{bmatrix}\text{x:plurality}(\textit{Rec})\\ \text{e:passenger\_frame\_pl(x)}\end{bmatrix} .\ \left[\text{e:get\_a\_hot\_meal\_frame\_pl}(r.\text{x})\right]\urcorner)$$

Let us instantiate (157b) bit by bit with (158), assuming that the situation we are checking is s. We first compute the fixed point type of the foreground of the first argument. That is, (159a) which is identical to (159b).

(159) a. $\mathcal{F}(\lambda r{:}\left[\text{x:plurality}(\textit{Rec})\right] .$
$\left[\text{e:passenger\_frame\_pl}(r.\text{x})\right])$

b. $\begin{bmatrix}\text{x} & : & \text{plurality}(\textit{Rec})\\ \text{e} & : & \text{passenger\_frame\_pl(x)}\end{bmatrix}$

Then we compute the result of restricting the second argument to s, that is, $Q \restriction s$. This is given in (160).

(160) $\lambda r{:}\begin{bmatrix}\text{x:plurality}(\textit{Rec}) \\ \text{e:passenger\_frame\_pl(x)}\end{bmatrix} . \begin{bmatrix}\text{e}\underline{\varepsilon}s\text{:get\_a\_hot\_meal\_frame\_pl(}r\text{.x)}\end{bmatrix}$

We then compute the fixed point type of (160), given in (161a) which is identical with (161b) and equivalent to (161c).

(161) a. $\mathcal{F}(\lambda r{:}\begin{bmatrix}\text{x:plurality}(\textit{Rec}) \\ \text{e:passenger\_frame\_pl(x)}\end{bmatrix} . \begin{bmatrix}\text{e}\underline{\varepsilon}s\text{:get\_a\_hot\_meal\_frame\_pl(}r\text{.x)}\end{bmatrix})$

b. $\begin{bmatrix}\text{x:plurality}(\textit{Rec}) \\ \text{e:passenger\_frame\_pl(x)}\wedge(\text{get\_a\_hot\_meal\_frame\_pl(x)}\restriction s)\end{bmatrix}$

c. $\begin{bmatrix}\text{x:plurality}(\textit{Rec}) \\ \text{e}\underline{\varepsilon}s\text{:passenger\_frame\_pl(x)}\wedge\text{get\_a\_hot\_meal\_frame\_pl(x)}\end{bmatrix}$

In order to understand why (161b) and (161c) are equivalent we need to consider that the equivalences in (162) hold for any type system, $\mathbb{T}$.

(162) $T_1 \wedge (T_2 \restriction s) \approx_{\mathbb{T}} (T_1 \wedge T_2) \restriction s \approx_{\mathbb{T}} (T_1 \restriction s) \wedge T_2$

In addition we have the distribution equivalence of restriction as in (163).

(163) $(T_1 \wedge T_2) \restriction s \approx_{\mathbb{T}} (T_1 \restriction s) \wedge (T_2 \restriction s)$

These equivalences follow trivially from the witness condition for restricted types.

The final step specified in (157) involves merging (163b) with (164a), that is, (164b) which is identical with (164c).

(164) a. $\begin{bmatrix}\text{x} & : & \text{card(2000)}\end{bmatrix}$

b. $\begin{bmatrix}\text{x:plurality}(\textit{Rec}) \\ \text{e}\underline{\varepsilon}s\text{:passenger\_frame\_pl(x)}\wedge\text{get\_a\_hot\_meal\_frame\_pl(x)}\end{bmatrix} \underset{.}{\wedge} \begin{bmatrix}\text{x} & : & \text{card(2000)}\end{bmatrix}$

c. $\begin{bmatrix} \text{x:}\mathrm{plurality}(\textit{Rec})\wedge\text{card(2000)} \\ \text{e}\underline{\varepsilon}s\text{:passenger\_frame\_pl(x)}\wedge\text{get\_a\_hot\_meal\_frame\_pl(x)} \end{bmatrix}$

It is thus the type (164c) which is required to be non-empty by the content of an utterance of *2000 passengers get a hot meal*. This means that it is required that there is a plurality of records with cardinality 2000 where each record in the plurality is a passenger frame and the passenger in each frame gets a hot meal, or slightly more colloquially, there are 2000 separate events of a passenger getting a hot meal.

5.8 Summary of resources introduced

Items that are new since Chapter 4 are marked "**New!**" and items that have been revised since Chapter 4 are marked "**Revised!**".

5.8.1 Universal grammar resources

5.8.1.1 Types

Loc **Revised!** — $\begin{bmatrix} \text{x-coord} & : & \textit{Real} \\ \text{y-coord} & : & \textit{Real} \\ \text{z-coord} & : & \textit{Real} \end{bmatrix}$

Phon — a basic type

e : *Phon* iff e is a phonological event

SEvent — $\begin{bmatrix} \text{e-loc} & : & \textit{Loc} \\ \text{sp} & : & \textit{Ind} \\ \text{au} & : & \textit{Ind} \\ \text{e} & : & \textit{Phon} \\ \text{c}_{\text{loc}} & : & \text{loc(e,e-loc)} \\ \text{c}_{\text{sp}} & : & \text{speaker(e,sp)} \\ \text{c}_{\text{au}} & : & \text{audience(e,au)} \end{bmatrix}$ (as in Chapter 2)

Assgnmnt — a basic type

r : *Assgnmnt* iff r : *Rec* and $\mathrm{labels}(r) \subset \{\mathrm{x}_0, \mathrm{x}_1, \ldots\}$

PropCntxt — a basic type

r : *PropCntxt* iff r : *Rec* and $\mathrm{labels}(r) \cap \{\mathrm{x}_0, \mathrm{x}_1, \ldots\} = \emptyset$

Cntxt — $\begin{bmatrix} \mathfrak{s} & : & \textit{Assgnmnt} \\ \mathfrak{c} & : & \textit{PropCntxt} \end{bmatrix}$

CntxtType — a basic type

T : *CntxtType* iff $T \sqsubseteq$ *Cntxt*

xType **New!** — a basic type

T : *xType* iff T : *RecType* and x $\in \mathrm{labels}(T)$

Ppty **Revised!** — $\left[\begin{array}{lll} \text{bg} & : & \textit{xType} \\ \text{fg} & : & (\text{bg}{\rightarrow}\textit{RecType}) \end{array}\right]$

PlPpty **New!** — a basic type

P : *PlPpty* iff P : *Ppty* and for some type T, P.bg $\sqsubseteq \left[\text{x:}\mathrm{plurality}(T)\right]$

PPpty — $\left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}{\rightarrow}\textit{Ppty}) \end{array}\right]$

Quant — (*Ppty*→*RecType*)

PQuant — $\left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}{\rightarrow}\textit{Quant}) \end{array}\right]$

QuantDet — (*Ppty*→*Quant*)

PQuantDet — $\left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}{\rightarrow}\textit{QuantDet}) \end{array}\right]$

PRecType — $\left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}{\rightarrow}\textit{RecType}) \end{array}\right]$

Cont — *PRecType*∨*PPpty*∨*PQuant*∨*PQuantDet*

Cat — a basic type

s, np, det, n, v, vp : *Cat*

Syn — $\left[\begin{array}{lll} \text{cat} & : & \textit{Cat} \\ \text{daughters} & : & \textit{Sign}^{*} \end{array}\right]$

Sign — a basic type

σ : *Sign* iff σ : $\left[\begin{array}{lll} \text{s-event} & : & \textit{SEvent} \\ \text{syn} & : & \textit{Syn} \\ \text{cont} & : & \textit{Cont} \end{array}\right]$

SignType — a basic type

T : *SignType* iff $T \sqsubseteq$ *Sign*

S — $\left[\begin{array}{l} ^{\textit{Sign}} \\ \text{syn:}\left[\text{cat=s:}\textit{Cat}\right] \end{array}\right]$

NP — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=np:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

Det — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=det:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

N — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=n:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

V — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=v:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

VP — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=vp:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

NoDaughters — $\begin{bmatrix}\text{syn:}\begin{bmatrix}\text{daughters=}\varepsilon\text{:}\textit{Sign}^*\end{bmatrix}\end{bmatrix}$

Real **New!** — a basic type

n : *Real* iff n is a real number

Card **New!** — a basic type

n : *Card* iff n is a cardinal number (natural numbers with the addition of $\aleph_0, \aleph_1, \ldots$)

AmbTempFrame **New!** — $\begin{bmatrix} \text{x} & : & \textit{Real} \\ \text{loc} & : & \textit{Loc} \\ \text{e} & : & \text{temp(loc, x)} \end{bmatrix}$

TempRiseEventCntxt **New!** — $\begin{bmatrix} \text{fix} & : & \begin{bmatrix} \text{loc} & : & \textit{Loc} \end{bmatrix} \\ \text{scale} & : & (\textit{AmbTempFrame} \rightarrow \textit{Real}) \end{bmatrix}$

TempRiseEvent **New!** —

$$\lambda r{:}\textit{TempRiseEventCntxt}\ .$$
$$\begin{bmatrix} \text{e} & : & (\textit{AmbTempFrame}\|r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale(e[0])} < r.\text{scale(e[1])} \end{bmatrix}$$

PriceFrame **New!** — $\begin{bmatrix} \text{x} & : & \textit{Real} \\ \text{loc} & : & \textit{Loc} \\ \text{commodity} & : & \textit{Ind} \\ \text{e} & : & \text{price(commodity, loc, x)} \end{bmatrix}$

PriceRiseEventCntxt **New!** — $\begin{bmatrix} \text{fix} & : & \begin{bmatrix} \text{loc} & : & \textit{Loc} \\ \text{commodity} & : & \textit{Ind} \end{bmatrix} \\ \text{scale} & : & (\textit{PriceFrame} \rightarrow \textit{Real}) \end{bmatrix}$

PriceRiseEvent **New!** —

$$\lambda r{:}\textit{TempRiseEventCntxt}\ .\ \left[\begin{array}{lll} \text{e} & : & (\textit{PriceFrame}\|r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1]) \end{array}\right]$$

LocFrame **New!** — $\left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{loc} & : & \textit{Loc} \\ \text{e} & : & \text{at(x, loc)} \end{array}\right]$

LocRiseEventCntxt **New!** — $\left[\begin{array}{lll} \text{fix} & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \end{array}\right] \\ \text{scale} & : & (\textit{LocFrame} \rightarrow \textit{Real}) \end{array}\right]$

LocRiseEvent **New!** —

$$\lambda r{:}\textit{LocRiseEventCntxt}\ .\ \left[\begin{array}{lll} \text{e} & : & (\textit{LocFrame}\|r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1]) \end{array}\right]$$

5.8.1.2 Predicates

with arity $\langle$*Phon*, *Loc*$\rangle$

loc — e : loc(u, l) iff u is located at l in e

with arity $\langle$*Phon*, *Ind*$\rangle$

speaker — e : speaker(u, a) iff u is the speaker of u in e

audience — e : audience(u, a) iff u is the audience of u in e

with arity $\langle$*Card*$\rangle$

card **New!** — X : card(n) iff for some T, $X : \text{set}(T)$ and $|X| = n$

card_at_least **New!** — X : card_at_least(n) iff for some T, $X : \text{set}(T)$ and $|X| \geq n$

card_at_most **New!** — X : card_at_most(n) iff for some T, $X : \text{set}(T)$ and $|X| \leq n$

with arity $\langle$*Ppty*$\rangle$

unique **New!** — s : unique(P) iff $|\,[{\downarrow}P \restriction s]\,| = 1$

with arity $\langle$*Ppty*, *Ppty*$\rangle$

exist — s : exist(P,Q) iff $[{\downarrow}P] \cap [{\downarrow}Q \restriction s] \neq \emptyset$

every **New!** — s : every(P,Q) iff $[{\downarrow}P] \subseteq [{\downarrow}Q \restriction s]$

with arity $\langle \textit{PlPpty}, \textit{PlPpty}\rangle$

exactly_n **New!** — for n a natural number,
s : exactly_$n(P, Q)$ iff s : at_least_$n(P, Q)\wedge$at_most_$n(P, Q)$

at_least_n **New!** — for n a natural number,
s : at_least_$n(P, Q)$ iff $\lceil\mathcal{F}((Q\restriction s).\text{fg}\,|_{\mathcal{F}(P.\text{fg})}) \mathbin{\dot\wedge} \left[\text{x:card\_at\_least}(n)\right]\rceil \neq \emptyset$

at_most_n **New!** — for n a natural number,
s : at_most_$n(P, Q)$ iff $r : \mathcal{F}((Q\restriction s).\text{fg}\,|_{\mathcal{F}(P.\text{fg})})$ implies $r : \left[\text{x:card\_at\_most}(n)\right]$

with arity $\{\langle T\rangle \mid T \text{ is a type}\}$

be — e : be(a) iff $a\varepsilon e$

with arity $\langle \textit{Loc}, \textit{Real}\rangle$

temp **New!** — e : temp(l, n) iff n is the temperature at l in e.

with arity $\langle \textit{Real}, \textit{Real}\rangle$

less-than **New!** — e : less-than(n, m) iff $n\varepsilon e$, $m\varepsilon e$ and $n < m$

5.8.1.3 Scales

ζ_{temp} **New!** — $\lambda r{:}\textit{AmbTempFrame}\ .\ r.\text{x} : (\textit{AmbTempFrame} \to \textit{Real})$

ζ_{height} **New!** — $\lambda r{:}\textit{LocFrame}\ .\ r.\text{loc.z-coord} : (\textit{LocFrame} \to \textit{Real})$

ζ_{age} **New!** — $\lambda r{:}\begin{bmatrix}\text{x:}\textit{Ind}\\ \text{age:}\textit{Real}\\ \text{c}_{\text{age}}\text{:age\_of(x,age)}\end{bmatrix}\ .\ r.\text{age} : (\begin{bmatrix}\text{x:}\textit{Ind}\\ \text{age:}\textit{Real}\\ \text{c}_{\text{age}}\text{:age\_of(x,age)}\end{bmatrix} \to \textit{Real})$

5.8.1.4 Lexicon

Lex

If T_{phon} is a phonological type (that is, $T_{\text{phon}} \sqsubseteq \textit{Phon}$) and T_{sign} is a sign type (that is, $T_{\text{sign}} \sqsubseteq \textit{Sign}$), then we shall use Lex$(T_{\text{phon}}, T_{\text{sign}})$ to represent

$((T_{\text{sign}} \mathbin{\dot\wedge} \left[\text{s-event:}\left[\text{e:}T_{\text{phon}}\right]\right]) \mathbin{\dot\wedge} \textit{NoDaughters})$

SemCommonNoun(T_{bg}, p) **Revised!**

If p is a predicate with arity $\langle \textit{Ind}\rangle$ and T_{bg} is a type (of context), then SemCommonNoun(T_{bg}, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\begin{bmatrix}\text{x}{:}\textit{Ind}\end{bmatrix}\ .\ \begin{bmatrix}\text{e} & : & p(r.\text{x})\end{bmatrix}\urcorner\urcorner$$

If p is a predicate with arity $\langle \textit{Rec}\rangle$ and T_{bg} is a type (of context), then SemCommonNoun(T_{bg}, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\begin{bmatrix}\text{x}{:}\textit{Rec}\end{bmatrix}\ .\ \begin{bmatrix}\text{e} & : & p(r.\text{x})\end{bmatrix}\urcorner\urcorner$$

$\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$ **Revised!**

If T_{phon} is a phonological type, p is a predicate with arity $\langle \textit{Ind}\rangle$ or $\langle \textit{Rec}\rangle$ and T_{bg} is a type (of context), then $\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$ is

$$\text{Lex}(T_{\text{phon}}, \textit{N}) \wedge\!\!\!\!\cdot\ \begin{bmatrix}\text{cont=SemCommonNoun}(T_{\text{bg}}, p){:}\textit{PPpty}\end{bmatrix}$$

SemPropName(T_{phon})

If T_{phon} is a phonological type, then SemPropName(T_{phon}) is

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix}\text{x}{:}\textit{Ind} \\ \text{e:named}(\text{x}, T_{\text{phon}})\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(c.\mathfrak{c})\urcorner$$

$\text{Lex}_{\text{PropName}}(T_{\text{phon}}\ )$

If T_{phon} is a phonological type,

then $\text{Lex}_{\text{PropName}}(T_{\text{phon}})$ is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \wedge\!\!\!\!\cdot\ \begin{bmatrix}\text{cnt=SemPropName}(T_{\text{phon}}){:}\textit{PQuant}\end{bmatrix}$$

SemPron

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt} \\ \mathfrak{s}{:}\begin{bmatrix}\text{x}_0{:}\textit{Ind}\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(\begin{bmatrix}\text{x}{=}c.\mathfrak{s}.\text{x}_0\end{bmatrix})\urcorner$$

LexPron(T_{phon})

If T_{phon} is a phonological type, then LexPron(T_{phon}) is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \wedge\!\!\!\!\cdot\ \begin{bmatrix}\text{cont=SemPron}{:}\textit{PQuant}\end{bmatrix}$$

SemNumeral(n) **New!**

If n is a real number, then SemNumeral(n) is

$$\ulcorner\lambda c{:}\textit{Cntxt}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(\begin{bmatrix}\text{x}{=}n\end{bmatrix})\urcorner$$

$\text{Lex}_{\text{numeral}}(T_{\text{phon}}, n)$ **New!**

If T_{phon} is a phonological type and n is a real number, then $\text{Lex}_{\text{numeral}}(T_{\text{phon}}, n)$ is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \wedge\!\!\!\!\cdot\ \begin{bmatrix}\text{cnt=SemNumeral}(n){:}\textit{PQuant}\end{bmatrix}$$

SemIndefArt **Revised!**

$$\lambda Q{:}\mathit{Ppty}\ .\ \ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q & : & \mathit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{bmatrix}\urcorner$$

$\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ **Revised!**

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{Det}) \mathbin{\dot{\wedge}} \big[\text{cont}{=}\text{SemIndefArt}{:}(\mathit{Ppty}{\rightarrow}\mathit{PQuant})\big]$$

SemUniversal **New!**

$$\lambda Q{:}\mathit{Ppty}\ .\ \ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q & : & \mathit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty} \\ \text{e} & : & \text{every(restr, scope)} \end{bmatrix}\urcorner$$

$\text{Lex}_{\text{Universal}}(T_{\text{Phon}})$ **New!**

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{Universal}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{Det}) \mathbin{\dot{\wedge}} \big[\text{cont}{=}\text{SemUniversal}{:}(\mathit{Ppty}{\rightarrow}\mathit{PQuant})\big]$$

SemDefArt **New!**

$$\lambda Q{:}\mathit{Ppty}\ .\ \ulcorner\lambda c{:}\begin{bmatrix} \mathit{Cntxt} \\ \mathfrak{c}{:}\big[\text{e}{:}\text{unique}(Q)\big] \end{bmatrix}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q\restriction c.\mathfrak{c}.\text{e} & : & \mathit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty} \\ \text{e} & : & \text{every(restr, scope)} \end{bmatrix}\urcorner$$

$\text{Lex}_{\text{DefArt}}(T_{\text{Phon}})$ **New!**

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{Det}) \mathbin{\dot{\wedge}} \big[\text{cont}{=}\text{SemDefArt}{:}(\mathit{Ppty}{\rightarrow}\mathit{PQuant})\big]$$

$\text{SemIntransVerb}(T_{\text{bg}}, p)$ **Revised!**

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}\rangle$, then $\text{SemIntransVerb}(T_{\text{bg}}, p)$ is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \big[\ \text{e}\ :\ p(r.\text{x})\ \big]\urcorner\urcorner$$

If $T_{\text{bg}} \sqsubseteq \left[\mathfrak{c}:\mathit{Rec}\right]$ is a record type (for context) and p is a predicate with arity $\langle \mathit{Rec}, \mathit{Rec}\rangle$, then SemIntransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\left[\text{x}{:}\mathit{Rec}\right]\ .\ \left[\ \text{e} \ :\ p(r.\text{x},\, c.\mathfrak{c})\ \right]\urcorner\urcorner$$

Lex$_{\text{IntransVerb}}$(T_{phon}, T_{bg}, p) **Revised!**

If T_{phon} is a phonological type, $T_{\text{bg}} \sqsubseteq \left[\mathfrak{c}:\mathit{Rec}\right]$ a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}\rangle$ or $\langle \mathit{Rec}, \mathit{Rec}\rangle$, then Lex$_{\text{IntransVerb}}$($T_{\text{phon}}$, T_{bg}, p) is

$$\text{Lex}(T_{\text{phon}},\ V_i) \mathbin{\dot{\wedge}} \left[\text{cnt=SemIntransVerb}(T_{\text{bg}},\, p){:}\mathit{PPpty}\right]$$

SemTransVerb(T_{bg}, p) **Revised!**

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}, \mathit{Ind}\rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \lambda \mathcal{Q}{:}\mathit{Quant}\ .\ \ulcorner\lambda r_1{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \mathcal{Q}(\ulcorner\lambda r_2{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \left[\ \text{e}\ :\ p(r_1.\text{x},\, r_2.\text{x})\ \right]\urcorner)\urcorner\urcorner$$

Lex$_{\text{TransVerb}}$(T_{phon}, T_{bg}, p)

If T_{phon} is a phonological type, T_{bg} a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}, \mathit{Ind}\rangle$, then Lex$_{\text{TransVerb}}$($T_{\text{phon}}$, T_{bg}, p) is

$$\text{Lex}(T_{\text{phon}},\ V_t) \mathbin{\dot{\wedge}} \left[\text{cnt=SemTransVerb}(T_{\text{bg}},\, p){:}\mathit{PPpty}\right]$$

SemBe **Revised!**

SemBe$_{\text{ID}}$

$$\begin{array}{l}\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\left[\text{ty}{:}\mathit{Type}\right]\end{bmatrix}\ .\\ \quad\lambda\mathcal{Q}{:}\mathit{Quant}\ .\\ \quad\quad\ulcorner\lambda r_1{:}\left[\text{x}{:}c.\mathfrak{c}.\text{ty}\right]\ .\\ \quad\quad\quad\mathcal{Q}(\ulcorner\lambda r_2{:}\left[\text{x}{:}c.\mathfrak{c}.\text{ty}\right]\ .\ \begin{bmatrix}\text{x=}r_1.\text{x},\ r_2.\text{x} & : & c.\mathfrak{c}.\text{ty}\\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner)\urcorner\urcorner\end{array}$$

SemBe$_{\text{scalar}}$

$$\begin{array}{l}\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{ty}{:}\mathit{Type}\\ \text{sc}{:}(\text{ty}\to\mathit{Real})\end{bmatrix}\end{bmatrix}\ .\\ \quad\lambda\mathcal{Q}{:}\mathit{Quant}\ .\\ \quad\quad\ulcorner\lambda r_1{:}\left[\text{x}{:}c.\mathfrak{c}.\text{ty}\right]\ .\\ \quad\quad\quad\mathcal{Q}(\ulcorner\lambda r_2{:}\left[\text{x}{:}\mathit{Real}\right]\ .\ \begin{bmatrix}\text{x=}c.\mathfrak{c}.\text{sc}(r_1.\text{x}),\ r_2.\text{x} & : & \mathit{Real}\\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner)\urcorner\urcorner\end{array}$$

Lex$_{\text{be}}$(T_{Phon}) **Revised!**

If T_{Phon} is a phonological type, then Lex$_{\text{be}_{\text{ID}}}$(T_{Phon}) is

$$\text{Lex}(T_{\text{Phon}},\ V) \mathbin{\dot{\wedge}} \left[\text{cont=SemBe}_{\text{ID}}{:}(\mathit{Quant}\to\mathit{Ppty})\right]$$

If T_{Phon} is a phonological type, then $\mathrm{Lex}_{\mathrm{be}_{\mathrm{scalar}}}(T_{\mathrm{Phon}})$ is

$$\mathrm{Lex}(T_{\mathrm{Phon}}, V) \mathbin{\dot\wedge} \left[\text{cont=SemBe}_{\text{scalar}}\text{:}(\textit{Quant}\rightarrow\textit{Ppty})\right]$$

$\mathrm{FrameType}(p)$ **New!**

$\mathrm{FrameType}$ is a partial function on predicates, p, with arity $\langle \textit{Ind} \rangle$ which can be defined for particular agents and particular times, which obeys the constraint:

$$\mathrm{FrameType}(p) \sqsubseteq \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & p(\text{x}) \end{array}\right]$$

p_frame **New!**

1. If p is a predicate in the domain of $\mathrm{FrameType}$, then p_frame is a predicate with arity $\langle \textit{Rec} \rangle$.
2. e : p_frame(r) iff r : $\mathrm{FrameType}(p)$ and $e = r$

p_pl **New!**

1. If p is a singular predicate (i.e. there is no p' such that $p = p'$_pl) with arity $\langle T \rangle$, then p_pl is a predicate with arity $\langle \mathrm{plurality}(T) \rangle$
2. e : p_pl(A) if for all $a \in A$, e : $p(a)$

CommonNounIndToFrame **New!**

If T_{phon} is a phonological type, p is a predicate with arity $\langle \textit{Ind} \rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then
CommonNounIndToFrame($\mathrm{Lex}_{\mathrm{CommonNoun}}(T_{\mathrm{phon}}, T_{\mathrm{bg}}, p)$) =

$$\mathrm{Lex}_{\mathrm{CommonNoun}}(T_{\mathrm{phon}}, T_{\mathrm{bg}}, p\text{\_frame})$$

RestrictCommonNoun **New!**

If T_{phon} is a phonological type, p is a predicate, T_{bg} and T_{res} are record types and Σ is $\mathrm{Lex}_{\mathrm{CommonNoun}}(T_{\mathrm{phon}}, T_{\mathrm{bg}}, p)$, then RestrictCommonNoun(Σ, T_{res}) is

$$\Sigma \mathbin{\boxed{\dot\wedge}} \left[\ \text{cont=}\ulcorner\lambda c{:}T_{\mathrm{bg}}\ .\ \ulcorner\text{SemCommonNoun}(T_{\mathrm{bg}}, p)(c)\,|_{T_{\mathrm{res}}}\urcorner\urcorner \quad : \quad \textit{PPpty}\ \right]$$

IntransVerbIndToFrame **New!**

If T_{phon} is a phonological type, p is a predicate with arity $\langle \textit{Ind} \rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then
IntransVerbIndToFrame($\mathrm{Lex}_{\mathrm{IntransVerb}}(T_{\mathrm{phon}}, T_{\mathrm{bg}}, p)$) =

$$\mathrm{Lex}_{\mathrm{IntransVerb}}(T_{\mathrm{phon}}, T_{\mathrm{bg}}, p\text{\_frame})$$

PluralCommonNoun **New!**

We assume that 'pluralnoun' is a function that maps phonological types for singular common nouns to corresponding phonological types for plural common nouns.

If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T \rangle$ and T_{bg} is a record type then PluralCommonNoun($\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$$\text{Lex}_{\text{CommonNoun}}(\text{pluralnoun}(T_{\text{phon}}), T_{\text{bg}}, p\text{\_pl})$$

PluralIntransVerb **New!**

We assume that 'pluralverb' is a function that maps phonological types for singular verbs to corresponding phonological types for plural verbs.

If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T \rangle$ and T_{bg} is a record type then PluralIntransVerb($\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$$\text{Lex}_{\text{IntransVerb}}(\text{pluralverb}(T_{\text{phon}}), T_{\text{bg}}, p\text{\_pl})$$

5.8.1.5 Constituent structure

RuleDaughters($T_{\text{daughters}}, T_{\text{mother}}$)

If T_{mother} is a sign type and $T_{\text{daughters}}$ is a type of strings of signs then

$$\text{RuleDaughters}(T_{\text{daughters}}, T_{\text{mother}})$$

is

$$\lambda u{:}T_{\text{daughters}} \,.\, T_{\text{mother}} \mathbin{\dot{\wedge}} \left[\text{syn}{:}\left[\text{daughters}{=}u{:}T_{\text{daughters}}\right]\right]$$

ConcatPhon

$$\lambda u{:}\left[\text{s-event}{:}\left[\text{e}{:}\textit{Phon}\right]\right]^{+} \,. \\ \left[\ \text{s-event} \ : \ \left[\ \text{e=concat}_i(u[i].\text{s-event.e}) \ : \ \textit{Phon}\ \right]\ \right]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$

If T_{mother} is a sign type and $T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$ are sign types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1} \ldots T_{\text{daughter}_n}$$

represents

$$\text{RuleDaughters}(T_{\text{mother}}, T_{\text{daughter}_1}{}^\frown \ldots {}^\frown T_{\text{daughter}_n}) \mathbin{\dot{\wedge}} \text{ConcatPhon}$$

$\alpha @ \beta$

If $\alpha : \begin{bmatrix} \text{bg}{:}\textit{CntxtType} \\ \text{fg}{:}(\text{bg}{\to}(T_1 \to T_2)) \end{bmatrix}$ and $\beta : \begin{bmatrix} \text{bg}{:}\textit{CntxtType} \\ \text{fg}{:}(\text{bg}{\to}\ T_1) \end{bmatrix}$ then the *combination of* α *and* β *based on functional application*, $\alpha @ \beta$, is

$$\ulcorner\lambda c{:}[\alpha.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}} \wedge \mathrm{incr}_{\mathfrak{s}.\mathrm{x}}([\beta.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}, \alpha.\mathrm{bg})\ .\ [\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}}(c)(\mathrm{incr}_{\mathfrak{s}.\mathrm{x}}([\beta.\mathrm{fg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}, \alpha.\mathrm{bg})(c))\urcorner$$

ContForwardApp(T_{arg}, T_{res})

If T_{arg} and T_{res} are types, then ContForwardApp(T_{arg}, T_{res}) is

$$\lambda u{:}\left[\mathrm{cont}{:}\begin{bmatrix}\mathrm{bg}{:}\mathit{CntxtType}\\ \mathrm{fg}{:}(\mathrm{bg}{\rightarrow}(T_{\mathrm{arg}} \to T_{\mathrm{res}}))\end{bmatrix}\right]^\frown\left[\mathrm{cont}{:}\begin{bmatrix}\mathrm{bg}{:}\mathit{CntxtType}\\ \mathrm{fg}{:}(\mathrm{bg}{\rightarrow}\ T_{\mathrm{arg}})\end{bmatrix}\right]\ .\ \left[\mathrm{cont}{=}u[0].\mathrm{cont}@u[1].\mathrm{cont}{:}\begin{bmatrix}\mathrm{bg}{:}\mathit{CntxtType}\\ \mathrm{fg}{:}(\mathrm{bg}{\rightarrow}\ T_{\mathrm{res}})\end{bmatrix}\right]$$

$T_{\mathrm{mother}} \longrightarrow T_{\mathrm{daughter}_1}\ T_{\mathrm{daughter}_2}\ \mid\ T'_{\mathrm{daughter}_1}(T'_{\mathrm{daughter}_2} : T_{\mathrm{arg}}) : T_{\mathrm{res}}$

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and T_{arg} and T_{res} are content types, then

$$T_{\mathrm{mother}} \longrightarrow T_{\mathrm{daughter}_1}\ T_{\mathrm{daughter}_2}\ \mid\ T'_{\mathrm{daughter}_1}(T'_{\mathrm{daughter}_2} : T_{\mathrm{arg}}) : T_{\mathrm{res}}$$

is

$$T_{\mathrm{mother}} \longrightarrow T_{\mathrm{daughter}_1}\ T_{\mathrm{daughter}_2} \wedge \mathrm{ContForwardApp}(T_{\mathrm{arg}}, T_{\mathrm{res}})$$

$\alpha@@\beta$ **New!**

If $\alpha : (T_1 \to \begin{bmatrix}\mathrm{bg}{:}\mathit{CntxtType}\\ \mathrm{fg}{:}(\mathrm{bg}{\rightarrow}\ T_2)\end{bmatrix})$ and $\beta : \begin{bmatrix}\mathrm{bg}{:}\mathit{CntxtType}\\ \mathrm{fg}{:}(\mathrm{bg}{\rightarrow}\ T_1)\end{bmatrix}$ then the *combination of* α *and* β *based on functional application*, $\alpha@@\beta$, is

$$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt} & & \\ \mathfrak{c} & : & \begin{bmatrix}\mathrm{s} & : & \beta.\mathrm{bg}\\ \mathrm{f} & : & \alpha(\beta(s)).\mathrm{bg}\\ \mathrm{a}{=}\mathrm{s}.\mathfrak{c} & : & \mathit{PropCntxt}\end{bmatrix}\\ \mathfrak{s}{=}\mathfrak{c}.\mathrm{s}.\mathfrak{s} & : & \mathit{Assgnmnt}\end{bmatrix}\ .\ [\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}}([\beta]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}(c))(c)\urcorner$$

ContForwardApp$_{@@}$(T_{arg}, T_{res}) **New!**

If T_{arg} and T_{res} are types, then ContForwardApp$_{@@}$(T_{arg}, T_{res}) is

$$\lambda u{:}\left[\mathrm{cont}{:}(T_{\mathrm{arg}} \to T_{\mathrm{res}})\right]^\frown\left[\mathrm{cont}{:}T_{\mathrm{arg}}\right]\ .\ \left[\mathrm{cont}{=}u[0].\mathrm{cont}@@u[1].\mathrm{cont}{:}T_{\mathrm{res}}\right]$$

$T_{\mathrm{mother}} \longrightarrow T_{\mathrm{daughter}_1}\ T_{\mathrm{daughter}_2}\ \mid\ T'_{\mathrm{daughter}_1}({}_{@@}T'_{\mathrm{daughter}_2} : T_{\mathrm{arg}}) : T_{\mathrm{res}}$ **New!**

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and T_{arg} and T_{res} are content types, then

$$T_{\mathrm{mother}} \longrightarrow T_{\mathrm{daughter}_1}\ T_{\mathrm{daughter}_2}\ \mid\ T'_{\mathrm{daughter}_1}({}_{@@}T'_{\mathrm{daughter}_2} : T_{\mathrm{arg}}) : T_{\mathrm{res}}$$

is

$$T_{\mathrm{mother}} \longrightarrow T_{\mathrm{daughter}_1}\ T_{\mathrm{daughter}_2} \wedge \mathrm{ContForwardApp}_{@@}(T_{\mathrm{arg}}, T_{\mathrm{res}})$$

5.8.1.6 Action rules

(as in Chapter 3)

5.8.2 Universal speech act resources

(as in Chapter 2)

5.8.3 Universal dialogue resources

(as in Chapter 4)

5.8.4 English resources

5.8.4.1 Types and predicates

Basic phonological types for words

{"Dudamel", "is", "a", "conductor", "Beethoven", "composer", "Uchida", "pianist", "aha", "ok", "leaves" , "hugs", "dog" **New!**, "nine" **New!**, "ninety" **New!**}

Predicates

with arity $\langle \mathit{Ind} \rangle$ {conductor, composer, pianist, leave, dog **New!**, passenger **New!**}

with arity $\langle \mathit{Ind}, \mathit{Ind} \rangle$ {hug}

with arity $\langle \mathit{Rec}, \mathit{Rec} \rangle$ — {rise **New!**}

e : rise(r, c) if

r : *AmbTempFrame*,
c : *TempRiseEventCntxt* and
e : $\mathit{TempRiseEvent}(c) \mathbin{\dot{\wedge}} \left[\mathrm{e}{:}\left[\mathrm{t}_0{=}r{:}\mathit{AmbTempFrame}\right]\right]$

or if

r : *PriceFrame*,
c : *PriceRiseEventCntxt* and
e : $\mathit{PriceRiseEvent}(c) \mathbin{\dot{\wedge}} \left[\mathrm{e}{:}\left[\mathrm{t}_0{=}r{:}\mathit{PriceFrame}\right]\right]$

or if

r : *LocFrame*,
c : *LocRiseEventCntxt* and
e : $\mathit{LocRiseEvent}(c) \mathbin{\dot{\wedge}} \left[\mathrm{e}{:}\left[\mathrm{t}_0{=}r{:}\mathit{LocFrame}\right]\right]$

with arity $\langle \mathit{Rec} \rangle$ — {temperature **New!**}

e : temperature(r) if

r : *AmbTempFrame* and $e = r$

with arity $\langle$*Ind*, *TravelFrame*$\rangle$ — {take_journey **New!**}
s : take_journey(a, e) iff $s = e$ and e.traveller = a

Frame types

$$\textit{DogFrame}\ \textbf{New!}\ \text{—}\ \begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \\ \text{age} & : & \textit{Real} \\ \text{c}_{\text{age}} & : & \text{age\_of(x,age)} \end{bmatrix}$$

$$\textit{TravelFrame}\ \textbf{New!}\ \text{—}\ \begin{bmatrix} \text{traveller} & : & \textit{Ind} \\ \text{source} & : & \textit{Loc} \\ \text{goal} & : & \textit{Loc} \end{bmatrix}$$

$$\textit{PassengerFrame}\ \textbf{New!}\ \text{—}\ \begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{passenger(x)} \\ \text{journey} & : & \textit{TravelFrame} \\ \text{c}_{\text{travel}} & : & \text{take\_journey(x, journey)} \end{bmatrix}$$

5.8.4.2 Grammar

Lexical sign types

{Lex$_{\text{PropName}}$(“Dudamel”),
Lex$_{\text{PropName}}$(“Beethoven”),
Lex$_{\text{Pron}}$(“he”),
Lex$_{\text{numeral}}$(“nine”, 9) **New!**,
Lex$_{\text{numeral}}$(“ninety”, 90) **New!**,
Lex$_{\text{IndefArt}}$(“a”),
Lex$_{\text{Universal}}$(“every”) **New!**,
Lex$_{\text{DefArt}}$(“the”) **New!**,
Lex$_{\text{CommonNoun}}$(“composer”, *Rec*, composer) **Revised!**,
Lex$_{\text{CommonNoun}}$(“conductor”, *Rec*, conductor) **Revised!**,
Lex$_{\text{CommonNoun}}$(“dog”, *Rec*, dog) (= $\Sigma_{\text{“dog”}}$) **New!**,
RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“dog”}}$), *DogFrame*) **New!**,
Lex$_{\text{CommonNoun}}$(“passenger”, *Rec*, passenger) (= $\Sigma_{\text{“passenger”}}$) **New!**,
RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“passenger”}}$), *PassengerFrame*) **New!**,
Lex$_{\text{CommonNoun}}$(“temperature”, *Rec*, temperature) (= $\Sigma_{\text{“temperature”}}$) **New!**,
RestrictCommonNoun($\Sigma_{\text{“temperature”}}$, *AmbTempFrame*) **New!**,
Lex$_{\text{IntransVerb}}$(“leave”, *Rec*, leave),
Lex$_{\text{IntransVerb}}$(“run”, *Rec*, run), **New!**,
Lex$_{\text{IntransVerb}}$(“rise”, $\begin{bmatrix}\text{c}:\textit{TempRiseEventCntxt}\end{bmatrix}$, rise) **New!**,
Lex$_{\text{IntransVerb}}$(“rise”, $\begin{bmatrix}\text{c}:\textit{PriceRiseEventCntxt}\end{bmatrix}$, rise) **New!**,

$\text{Lex}_{\text{IntransVerb}}$(“rise”, $\begin{bmatrix}\mathfrak{c}{:}\textit{LocRiseEventCntxt}\end{bmatrix}$, rise) **New!**,
$\text{Lex}_{\text{TransVerb}}$(“hug”, *Rec*, hug),
$\text{Lex}_{\text{be}_{\text{ID}}}$(“is”) **Revised!**,
$\text{Lex}_{\text{be}_{\text{scalar}}}$(“is”) **Revised!**,
Lex(“ok”, *S*),
Lex(“aha”, *S*) }

Constituent structure rule components

CnstrIsA

$$\lambda u{:}V \mathbin{\dot\wedge} \begin{bmatrix}\text{s-event:}\begin{bmatrix}\text{e:“is”}\end{bmatrix}\end{bmatrix} {}^\frown NP \mathbin{\dot\wedge} \begin{bmatrix}\text{syn:}\begin{bmatrix}\text{daughters:}Det \mathbin{\dot\wedge} \begin{bmatrix}\text{s-event:}\begin{bmatrix}\text{e:“a”}\end{bmatrix}\end{bmatrix} \\ {}^\frown N \mathbin{\dot\wedge} \begin{bmatrix}\text{cont:}Ppty\end{bmatrix}\end{bmatrix}\end{bmatrix}.$$
$$VP \mathbin{\dot\wedge} \begin{bmatrix}\text{cont=}u[2].\text{syn.daughters}[2].\text{cont:}Ppty\end{bmatrix}$$

Constituent structure rules

$\{S \longrightarrow NP\ VP \mid NP'(VP'{:}Ppty){:}RecType$,
$NP \longrightarrow Det\ N \mid Det'({}_{@@}N'{:}PPpty){:}PQuant$ **Revised!**,
$VP \longrightarrow V\ NP \mathbin{\dot\wedge}$ CnstrIsA,
$VP \longrightarrow V\ NP \mid V'(NP'{:}Quant){:}Ppty\}$

5.9 Summary

In this chapter we have proposed an analysis of frames as records which model situations (including events) and we have suggested that frame types (record types) are important in both the analysis of the Partee puzzle concerning rising temperatures and prices and in the analysis of quantification which involves counting events rather than individuals likes passengers or ships passing through a lock.

Our original inspiration for frames comes from the work of Fillmore (1982, 1985) and work on FrameNet (`https://framenet.icsi.berkeley.edu`). An important aspect of our approach to frames is that we treat them as first class objects. That is, they can be arguments to predicates and can be quantified over. While this is important, it is not surprising once we decide that frames are in fact situations (here modelled by records) or situation types (here modelled by record types). The distinction between frames and frame types is not made in the literature deriving from Fillmore's work but it seems to be an important distinction to draw if we wish to apply the notion of frame to the kind of examples we have discussed in this chapter.

The proposal that we have made for solving the Partee puzzle is closely related to the work of Löbner (2014, in prep) whose inspiration is from the work of Barsalou (1992b,a, 1999) rather than Fillmore. Barsalou's approach embedded in a theory of cognition based on perception and a conception of cognition as dynamic, that is, a system in a constant state of flux (Prinz and

Barsalou, 2014), seems much in agreement with what we are proposing in this book. Barsalou's (1999) characterization of basic frame properties constituting a frame as: "(1) predicates, (2) attribute-value bindings, (3) constraints, and (4) recursion" seem to have a strong family resemblance with our record types. Our proposal for incorporating frames into natural language semantics is, however, different from Löbner's in that he sees the introduction of a psychological approach based on frames as a reason to abandon a formal semantic approach whereas we see type theory as a way of combining the insights we have gained from model theoretic semantics with a psychologically oriented approach.

Our approach to frames has much in common with that of Kallmeyer and Osswald (2013) who use feature structures to characterize their semantic domain. We have purposely used record types in a way that makes them correspond both to feature structures and discourse representation structures which allows us to relate our approach to more traditional model theoretic semantics at the same time as being able to merge record types corresponding to unification in feature-based systems. However, our record types are included in a richer system of types including function types facilitates a treatment of quantification and binding which is not available in a system which treats feature structures as a semantic domain.[10]

[10]It is possible to code up a notation for quantification in feature structures but that is not the same as giving a semantics for it.

Chapter 6

Modality and intensionality without possible worlds

6.1 Possible worlds, modality and intensionality

Montague (1973) uses possible worlds to analyze both modality (represented in his fragment by the adverbs *possibly* and *necessarily*) and a variety of intensional constructions in addition to the temperature and price examples discussed in Chapter 5: intensional transitive verbs such as *seek*, intensional adverbs such as *voluntarily*, verbs of propositional attitudes such as *believe* and *assert* and verbs taking infinitival complements such as *try (to)* and *wish (to)*.

A short introduction to the use of possible worlds in modal logic and philosophical conceptions of possible worlds is given by Menzel (2015). As he points out at the beginning of this article possible worlds are considered to be totalities (or at least a limit) which include the situations which we are aware of around us.

The notion of possible world is intuitively appealing. We talk of living in the best (or worst) of all possible worlds. But equally we talk of the best (or worst) possibility. When we talk in such terms we normally have a small finite number of possibilities in mind which we are contrasting. This has led some authors to use the term “possible world” to refer not to a total universe but to a small set of facts that might obtain in some version of the world. This appears to be standard usage in probability theory (e.g. Halpern, 2003). It is important not to confuse this notion with the notion of possible world as a totality which is used in semantics, inherited from modal logic. This point is made by Cooper *et al.* (2014a) and Lappin (2015).

Problems have been raised for the notion possible world. These have to do with how you individuate and count them and how many possible worlds there must be. Rescher (1999) takes up these problems from a philosophical perspective. He argues that it is impossible to individuate possible worlds and therefore impossible to count them. Lappin (2015) takes up the representation problem for possible worlds. If you cannot represent possible worlds then you cannot individuate them. The central problem for possible worlds as they are talked about in the semantics literature seems to be that the intuitive way to distinguish one possible world from another is to find a proposition that is true in the first world but false in the second. This would be fine except that we now have the corresponding problem for propositions. Unfortunately the intuitive way of distinguishing between one proposition and another (if you are a possible worlds theorist) is to find a possible world in which the first proposition is true and the other is false. This, of course, is circular and will not give us an individuation of either possible worlds or propositions. The standard version of possible worlds semantics as proposed by Montague does not, of course, fall into this obvious trap. Worlds are not represented in terms of sets of propositions which are true in them. Rather we just define an interpretation to include a set of possible worlds and leave aside the question of how they have been individuated. In a sense it is fine from a technical point of view to have an arbitrary set whose membership we cannot represent as a central component of our semantic theory. But it leaves us with the suspicion that we are left with an abstract theory which we do not really know how to connect to any empirical observations of the world. If you take a mathematical view of the semantic enterprise as Montague did, this may be acceptable. But if you are interested in semantics as an aspect of human cognitive ability it can appear problematic. Traditional possible world semantics is a theory based on an assumed set of possible worlds. But it is not a theory of the possible worlds as such, beyond the claim that there is a set of them.

Despite this, there is an intuition about the set of possible worlds which possible world theorists hold onto: that they represent all the logical possibilities. This, at least, gives us a way of considering the required cardinality of the set of possible worlds. The issue of the cardinality of the set of possible worlds and its relationship to a psychological theory of language is something that is already taken up by Partee (1977). Here she refers to Lewis's (1973) argument that there must be at least $\beth_2$ (the cardinality of the power set of the power set of natural numbers) possible worlds. The argument[1] goes like this: suppose we have a family that goes on for ever. That is, there would be $\aleph_0$ members of the family. Now consider that in a logically possible world (though possibly not in biologically possible worlds) any subset of these family members might have blue eyes (none of them, all of them and all the possibilities in between). This gives us a set of possible worlds whose cardinality is the same as the power set of the natural numbers $2^{\aleph_0}$ or $\beth_1$, that is, the cardinality of the set of real numbers. Now consider the logical possibility that each of those possible worlds is biologically plausible. Again, logically speaking, any subset of those worlds could be biologically plausible. This will yield a set of possible worlds of cardinality $2^{2^{\aleph_0}}$ or $\beth_2$. In principle one could create sets of possible worlds of any of the infinitely many infinite cardinalities although as Lewis claims $\beth_2$ is probably sufficient for normal purposes.

Another argument for the uncountability of the set of possible worlds comes from usual assumptions about space and time. We normally assume that the set of moments of time has the same cardinality as the set of points on the real line, that is, that time is continuous. Similarly we

[1]which I first heard from Barbara Partee but for which I cannot find a published reference

also assume that space is continuous. Now for any possible world where an object is at a certain location at a certain time there is another logically possible world where that object is located at a different location or occupies its location in the first world at a different time. For each such world there are uncountably many different logically possible worlds in which the object is located elsewhere.

How do we manage to reason about such large numbers of possibilities? The answer we want to propose here is that we reason in terms of types. A single type has a set of witnesses and there are no constraints on the cardinality of the set of witnesses. Types which have infinitely many witnesses are not more complex than types which have a small finite number of witnesses. Reasoning with a type involves manipulating the structural object which is the type itself not the set of its witnesses. Thus, for example, reasoning with a record type may be more complex than reasoning with a basic type that has no components. But still a record type is always a finite structure and so we are not entering into the complexity of manipulating uncountable sets, even though the record type may be thought of as a "representation" for its set of witnesses which may indeed be an uncountable set. Here our approach connects with proof theoretic approaches. In proof theory we manipulate expressions in a language which may represent sets of objects. Our types are not expressions in a language but they are objects in our type theoretic universe which could be thought of as "representing" the set of their witnesses. This approach also makes it possible to have a learning theory where agents can be acquainted with a type without being acquainted with the complete set of its witnesses. Knowing a type whose witnesses are dogs does not mean that you are acquainted with the set of all dogs, but rather that you know a dog when you see one, that is, you have a reliable dog *classifier*. (See Larsson, 2013 for a discussion of the relationship between types and classifiers.) An important aspect of human cognitive processing is that it involves reasoning with the types themselves, treating them as first class citizens which can be arguments to predicates. This is what gives rise to modality and intensionality. Possibly this higher level reasoning is unique, or, at least, most fully developed, in humans.

We think of types like record types as being types of situations. If we want to keep to the idea of possible worlds as total universes it is straightforward to convert a type of situations, T, to a type of worlds, T^W, as long as we have a way of defining worlds as maximal situations. We could say that a world, w, is of type T^W just in case some part, s of w is of type T. Actually, we do not need to do this because of the way we have set up subtyping. If T is a record type and $s : T$, then if $s < s'$, that is s is a proper part of s' in the sense defined in Chapter 5, then $s' : T$. If we had a way of defining maximal situations, that is, situations s such that there is no s' such that $s < s'$, we could take these to be our worlds. The problem is, though that it is not clear that it is desirable, or even possible, to characterize a notion of maximal situation in this sense. Certainly, there is no notion of maximal record so our choice of modelling situations as records suggests that there is no notion of maximal situation. Our axioms say that given any record it is always possible to add a new field to it.[2]

[2]This fact is parallel to Proposition 2 in Barwise (1989), Ch. 8: *Every situation, s, is a proper part of some other situation, s'.*

6.2 Modal type systems

Let us develop further the story about Kim's walk in the park from Chapter 2. Kim continues her walk still thinking about the boy and the dog whom she had seen playing the game of fetch. She thinks, "Was the boy standing too close to the pond? Suppose he had fallen in. If he had been my son, I wouldn't have let him play just there." An important aspect of human cognition is that we are not only able to observe things as they are but also to conceive of alternatives which go beyond the completion of observed events in the way discussed in Chapter 2, Section 2.2. We can not only observe objects and perceive them to be of certain types we can also consider possibilities in which they belong to different types and perhaps do not belong to the type we have observed. We have managed to unhook type judgements from direct perception. While the seeds of this ability can be seen in the kind of event perception and prediction discussed in Chapter 2 in that it gives us a way to consider types which have not yet been realized, it is at least one step further in cognitive evolution to be able to consider alternative type assignments which do not correspond to completions of events already perceived.

This leads us to construct *modal type systems* with alternative assignments of objects to types.[3] Figure 6.1 provides an example of a modal system of basic types with two possibilities, one where the extensions of types T_1 and T_2 overlap and another possibility where they do not.

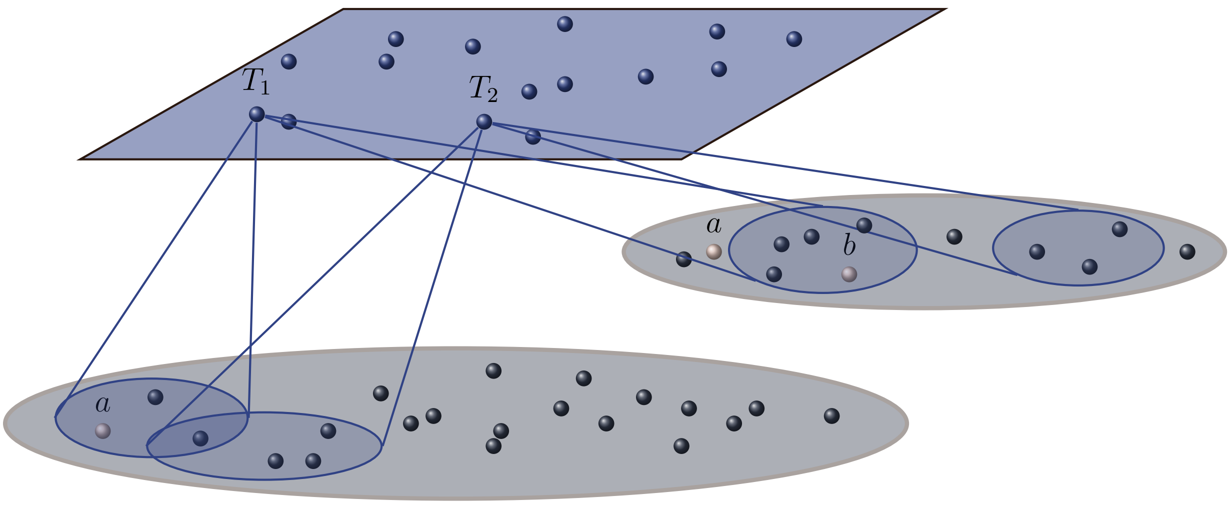


Figure 6.1: Modal system of basic types

The object a is of type T_1 in the first possibility but not in the second possibility. There is an object, b, of type T_1 in the second possibility. b does not exist at all in the first possibility. In the figure we just show two possibilities but our general definition in Appendix 9, introduced in Chapter 1, p. 40, allows for there to be any number of possibilities, including infinitely many.

[3] The term *modal* is taken from modal logic. See Hughes and Cresswell (1968) for a classic introduction. A modern introduction is to be found in Blackburn *et al.* (2001).

Given this apparatus we define four simple modal notions:

(necessary) equivalence Two types are (necessarily) equivalent just in case the extension of one type is identical with that of the other type in all the possibilities. While the different possibilities may provide different extensions for the types, it will always be the case that in any given possibility the two types will have the same extension.

subtype One type is a subtype of another just in case whatever possibility you look at it is always the case that the extension of the first type is a subset of the extension of the second. We can also say that the first type "entails" the second, that is, any object which is of the first type will also be of the second type, no matter which possibility you are considering.

necessity The notion of necessity we characterize for a type could be glossed as "necessarily realized" or "necessarily instantiated". A type will be necessary just in case there is something of the type in all the possibilities.

possibility This notion corresponds to "possibly realized" or "possibly instantiated". A type will be possible just in case there is some possibility according to which it has a non-null extension.

These notions are made precise for modal systems of complex types in (1) and (2), repeated in Appendix 9. As a preliminary we note that if

$$\mathbf{TYPE}_{MC} = \langle \mathbf{Type}_M, \mathbf{BType}, \langle \mathbf{PType}_M, \mathbf{Pred}, \mathbf{ArgIndices}, \mathit{Arity}\rangle, M\rangle_{M\in\mathcal{M}}$$

is a modal system of complex types based on $\mathcal{M}$, we shall use the notation $\mathbf{TYPE}_{MC_M}$ (where $M \in \mathcal{M}$) to refer to that system of complex types in $\mathbf{TYPE}_{MC}$ whose model is M. Let $\mathbf{Type}_{MC_{restr}}$ be $\bigcap\limits_{M\in\mathcal{M}} \mathbf{Type}_M$, the "restrictive" set of types which occur in all possibilities, and $\mathbf{Type}_{MC_{incl}}$ be $\bigcup\limits_{M\in\mathcal{M}} \mathbf{Type}_M$, the "inclusive" set of types which occur in at least one possibility. Then we can define modal notions either restrictively or inclusively (indicated by the subscripts r and i respectively).

(1) **restrictive modal notions**

a. for any $T_1, T_2 \in \mathbf{Type}_{MC_{restr}}$, T_1 *is (necessarily) equivalent*$_r$ *to* T_2 *in* $\mathbf{TYPE}_{MC}$, $T_1 \approx_{\mathbf{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$, $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_1\} = \{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_2\}$

b. for any $T_1, T_2 \in \mathbf{Type}_{MC_{restr}}$, T_1 *is a subtype*$_r$ *of* T_2 *in* $\mathbf{TYPE}_{MC}$, $T_1 \sqsubseteq_{\mathbf{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$, $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_1\} \subseteq \{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_2\}$

c. for any $T \in \textbf{Type}_{MC_{restr}}$, T *is necessary*$_r$ *in* $\textbf{TYPE}_{MC}$ iff for all $M \in \mathcal{M}$, $\{a \mid a :_{\textbf{TYPE}_{MC_M}} T\} \neq \emptyset$

d. for any $T \in \textbf{Type}_{MC_{restr}}$, T *is possible*$_r$ *in* $\textbf{TYPE}_{MC}$ iff for some $M \in \mathcal{M}$, $\{a \mid a :_{\textbf{TYPE}_{MC_M}} T\} \neq \emptyset$

(2) **inclusive modal notions**

a. for any $T_1, T_2 \in \textbf{Type}_{MC_{incl}}$, T_1 *is (necessarily) equivalent*$_i$ *to* T_2 *in* $\textbf{TYPE}_{MC}$, $T_1 \approx_{\textbf{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$, if T_1 and T_2 are members of $\textbf{Type}_M$, then $\{a \mid a :_{\textbf{TYPE}_{MC_M}} T_1\} = \{a \mid a :_{\textbf{TYPE}_{MC_M}} T_2\}$

b. for any $T_1, T_2 \in \textbf{Type}_{MC_{incl}}$, T_1 *is a subtype*$_i$ *of* T_2 *in* $\textbf{TYPE}_{MC}$, $T_1 \sqsubseteq_{\textbf{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$, if T_1 and T_2 are members of $\textbf{Type}_M$, then $\{a \mid a :_{\textbf{TYPE}_{MC_M}} T_1\} \subseteq \{a \mid a :_{\textbf{TYPE}_{MC_M}} T_2\}$

c. for any $T \in \textbf{Type}_{MC_{incl}}$, T *is necessary*$_i$ *in* $\textbf{TYPE}_{MC}$ iff for all $M \in \mathcal{M}$, if $T \in \textbf{Type}_M$, then $\{a \mid a :_{\textbf{TYPE}_{MC_M}} T\} \neq \emptyset$

d. for any $T \in \textbf{Type}_{MC_{incl}}$, T *is possible*$_i$ *in* $\textbf{TYPE}_{MC}$ iff for some $M \in \mathcal{M}$, if $T \in \textbf{Type}_M$, then $\{a \mid a :_{\textbf{TYPE}_{MC_M}} T\} \neq \emptyset$

It is easy to see that if any of the restrictive definitions holds for given types in a particular system then the corresponding inclusive definition will also hold for those types in that system.

This can be recast in terms of our informal proof theoretic notation where we use 'T true' to represent that T has some witness, '$T_1 \sqsubseteq_r T_2$' ('$T_1 \sqsubseteq_i T_2$') to represent that T_1 is a restrictive (inclusive) subtype of T_2, '$T_1 \approx_r T_2$' ('$T_1 \approx_i T_2$') to represent that T_1 is restrictively (inclusively) equivalent to T_2, 'T nec$_r$' ('T nec$_i$') to represent that T is restrictively (inclusively) necessary and 'T poss$_r$' ('T poss$_i$') to represent that T is restrictively (inclusively) possible. We give the restrictive notions in (3).

(3) **restrictive modal notions**

For $\mathcal{G}$ a modal system of complex types

a.
$$\frac{\begin{array}{c}[\Gamma \in \mathcal{G}]\\ \vdots \\ \Gamma \vdash T_1 \in \textbf{Type}\end{array} \quad \begin{array}{c}[\Gamma \in \mathcal{G}]\\ \vdots \\ \Gamma \vdash T_2 \in \textbf{Type}\end{array} \quad \begin{array}{c}[\Gamma \in \mathcal{G}, \Gamma \vdash a : T_1]\\ \vdots \\ \Gamma \vdash a : T_2\end{array}}{\mathcal{G} \vdash T_1 \sqsubseteq_r T_2}$$

b.
$$\frac{\mathcal{G} \vdash T_1 \sqsubseteq_r T_2 \quad \mathcal{G} \vdash T_2 \sqsubseteq_r T_1}{\mathcal{G} \vdash T_1 \approx_r T_2}$$

c.
$$\dfrac{\begin{array}{c}[\Gamma \in \mathcal{G}]\\ \vdots \\ \Gamma \vdash T \in \textbf{Type}\end{array} \quad \begin{array}{c}[\Gamma \in \mathcal{G}]\\ \vdots \\ \Gamma \vdash T \text{ true}\end{array}}{\mathcal{G} \vdash T \text{ nec}_r}$$

d.
$$\dfrac{\begin{array}{c}[\Gamma \in \mathcal{G}]\\ \vdots \\ \Gamma \vdash T \in \textbf{Type}\end{array} \quad \Gamma' \in \mathcal{G} \quad \Gamma' \vdash a : T}{\mathcal{G} \vdash T \text{ poss}_r}$$

The inclusive notions are given in (4).

(4) **inclusive modal notions**

For $\mathcal{G}$ a modal system of complex types

a.
$$\dfrac{\begin{array}{c}[\Gamma \in \mathcal{G}, \Gamma \vdash T_1 \in \textbf{Type}, \Gamma \vdash T_2 \in \textbf{Type}, \Gamma \vdash a : T_1]\\ \vdots \\ \Gamma \vdash a : T_2\end{array}}{\mathcal{G} \vdash T_1 \sqsubseteq_i T_2}$$

b.
$$\dfrac{\mathcal{G} \vdash T_1 \sqsubseteq_i T_2 \quad \mathcal{G} \vdash T_2 \sqsubseteq_i T_2}{\mathcal{G} \vdash T_1 \approx_i T_2}$$

c.
$$\dfrac{\Gamma' \in \mathcal{G} \quad \Gamma' \vdash T \in \textbf{Type} \quad \begin{array}{c}[\Gamma \in \mathcal{G}, \Gamma \vdash T \in \textbf{Type}]\\ \vdots \\ \Gamma \vdash T \text{ true}\end{array}}{\mathcal{G} \vdash T \text{ nec}_i}$$

d.
$$\dfrac{\Gamma \in \mathcal{G} \quad \Gamma \vdash T \in \textbf{Type} \quad \Gamma \vdash a : T}{\mathcal{G} \vdash T \text{ poss}_i}$$

Note that all of these notions are relativized to the modal system you are considering and the possibilities it offers. We may think of the family of assignments $\mathcal{A}$ as providing a modal base in the sense of Kratzer (2012). We may wish to consider very small families of assignments corresponding to the knowledge we have. Alternatively, we may want to consider strong logical variants of these modal notions where we consider all the logical possibilities, for example, all possible assignments of extensions to types.

6.3 Modality without possible worlds

Montague (1973) introduces *necessarily* and *possibly* as sentence adverbs, that is, they combine with a sentence to produce another sentence. If α is a sentence, then *necessarily* α is true in a possible world, w, just in case α is true in every possible world and *possibly* α is true in a possible world, w, just in case there is some possible world in which α is true.[4]

In Chapter 1, p. 40 and Appendix 9 we introduce modal type systems which are families of type systems, which we call *possibilities*, differing in their assignments of witnesses to basic types and ptypes. The important difference between possible worlds and possibilities is that for possibilities the parameters along which they can vary are fixed by the available types introduced in the type system, a well-defined notion, and one which varies depending on the particular type system. Thus we have a way of characterizing the dimensions along which the possibilities associated with a given type system vary and thus we have a way of representing the possibilities, whereas we do not have such a way of characterizing possible worlds. Building on the modal notions that we introduced in Section 6.2 we can introduce type constructors '$\Box$' and '$\Diamond$' corresponding to the operators in modal logic as in (5).

(5) If T is a type, then $\Box_r T$, $\Box_i T$, $\Diamond_r T$ and $\Diamond_i T$ are types

These types obey the constraints in (6) which correspond to the truth conditions for the corresponding operators in modal logic (S5).

(6) a. $\Box_{r/i} T$ is non-empty iff T is restrictively/inclusively necessary (approximately, non-empty in all possibilities)

b. $\Diamond_{r/i} T$ is non-empty iff T is restrictively/inclusively possible (approximately, non-empty in some possibility)

Making the witness conditions for these types meet the constraints in (6) is a little tricky. We do not have biconditionals, but two conditionals which correspond to introduction and elimination rules in proof theory. Suppose that $\mathbb{T}$ is a modal type system and that $p \in \mathbb{T}$ is a possibility in $\mathbb{T}$. Then for '$\Box_{r_1} T$ we have (7).

(7) a. If $a :_p \Box_{r/i} T$ then T is necessary$_{r/i}$ in $\mathbb{T}$

b. If T is necessary$_{r/i}$ in $\mathbb{T}$ then for any $p' \in \mathbb{T}$ there is some a' such that $a' :_{p'} \Box_{r/i} T$

---

[4]This simple treatment of modality corresponds to the modal logic system S5 where there is no restriction on accessibility between possible worlds (Hughes and Cresswell, 1968, 1996).

Note that the two clauses in (7) jointly entail (8).

(8) For any $p' \in \mathbb{T}$ there is some a' such that $a' :_{p'} \Box_{r/i}T$ ("$\Box_{r/i}T$ is true in p'") iff T is necessary$_{r/i}$ in $\mathbb{T}$

For '$\Diamond_{r/i}T$' we have (9).

(9) a. If $a :_p \Diamond_{r/i}T$ then T is possible$_{r/i}$ in $\mathbb{T}$
b. If T is possible$_{r/i}$ in $\mathbb{T}$ then for any $p' \in \mathbb{T}$ there is some a' such that $a' :_{p'} \Diamond_{r/i}T$

The two clauses in (9) jointly entail (10).

(10) For any $p' \in \mathbb{T}$ there is some a' such that $a' :_{p'} \Diamond_{r/i}T$ ("$\Diamond_{r/i}T$ is true in p'") iff T is possible$_{r/i}$ in $\mathbb{T}$

We can recreate the witness conditions in (7) and (9) in our informal proof theoretic representation as in (11) where we use $\mathcal{G} \vdash T$ nec/poss$_{r/i}$ to represent T is necessary/possible$_{r/i}$ in $\mathcal{G}$.

(11) For $\mathcal{G}$ a modal system of complex types:

a. $$\frac{\Gamma \in \mathcal{G} \qquad \Gamma \vdash a : \Box_{r/i}T}{\mathcal{G} \vdash T \text{ nec}_{r/i}}$$

b. $$\frac{\mathcal{G} \vdash T \text{ nec}_{r/i} \qquad \Gamma \in \mathcal{G}}{\Gamma \vdash \Box_{r/i}T \text{ true}}$$

c. $$\frac{\Gamma \in \mathcal{G} \qquad \Gamma \vdash a : \Diamond_{r/i}T}{\mathcal{G} \vdash T \text{ poss}_{r/i}}$$

d. $$\frac{\mathcal{G} \vdash T \text{ poss}_{r/i} \qquad \Gamma \in \mathcal{G}}{\Gamma \vdash \Diamond_{r/i}T \text{ true}}$$

This shows how we can, if we want to, recreate in TTR the simple S5 modal system (that is, where all possibilities are accessible to each other) that Montague (1973) uses.

We have used the symbols '$\Box$' and '$\Diamond$' above to suggest the relationship between our proposal in terms of types and the traditional formulas of modal logic. We could, however, have achieved the same effect as above by introducing predicates ‘$\mathrm{nec}_{r/i}$’ and ‘$\mathrm{poss}_{r/i}$’ with arity $\langle \mathit{Type} \rangle$ and giving ptypes of the form ‘$\mathrm{nec}_{r/i}(T)$’ and ‘$\mathrm{poss}_{r/i}(T)$’ the same witness conditions as above. We will pursue this option below where we will add additional arguments to the predicate.

How many possibilities are there in a modal type system? The answer to this question is that there can be as many as you choose for the given type system, ranging from a small finite number of possibilities to a higher order infinity. The definition of a modal type system given in Appendix 9 only requires that there be a family of possibilities. Thus this definition includes the kind of restricted sets of “possible worlds” differing along a small finite set of parameters which probability theorists talk of and indeed also linguistic semanticists talk of informally when they are in pedagogical explanatory mode (see, for example, Dowty *et al.*, 1981 and a lot of recent literature on inquisitive semantics such as Groenendijk and Roelofsen, 2012).

It is important in a modal type system that the identity criteria for the possibilities are determined by the types provided by the system. Two possibilities are distinct only if they differ in the witnesses associated with some basic type or ptype. It is not possible to make distinctions for which you do not have appropriate types available. Thus the range of possibilities is limited by the types which are available to classify objects.

This is not to say that we have eliminated all potential decidability problems from modal type systems. Of course, if the types that we use to construct the system are not decidable it may not be possible to decide on identity for possibilities. Even if all the types are guaranteed to be decidable, given an inifinite set of possibilities there cannot be any general guarantee that we can decide whether an arbitrary type is necessary or possible or not since we cannot visit every possibility in a finite amount of time. We can only be sure if we have some general argument about the possibilities which does not involve inspecting each possibility individually. But having a way of distinguishing between possibilities which may in the limit be undecidable is better than not having a way of distinguishing between possibilities, other than that they are distinct members of a set.

The work on modality in natural language which has followed after Montague’s original work all points to a more restricted kind of modality which involves arguing from some basic assumptions to a conclusion rather than considering all logical possibilities. This view of modality in natural language has been put forward by Kratzer in a body of work beginning with Kratzer (1977). This and other papers by Kratzer on modality are collected in revised and commented form in Kratzer (2012) and there is much other literature which builds on Kratzer’s ideas. An excellent introduction to Kratzer’s work is given in Chapter 3 of Portner (2009). The essential idea is that modals like *must* (corresponding to necessity) and *can* (corresponding to possibility) must be interpreted relative to a “conversational background” which in Kratzer (1981) (Chapter 2 of Kratzer, 2012) is split into two components, a *modal base* and an *ordering source*. The modal base is a set of propositions[5] which characterize the assumptions from which we are arguing. The ordering source is a set of propositions[6] which determine an ideal which we are trying to get as close to as possible. It is called an ordering source because Kratzer, following Lewis (1981), thinks of it as inducing a partial ordering on possible worlds, in terms of their closeness to the ideal. Kratzer's insight is that necessity and possibility in natural language should be defined relative to a modal base and an ordering source. In simple terms, a proposition, p, is necessary with respect to a modal base, b, and an ordering source (ideal), i, just in case p follows from the conjunction of b and i. A proposition, p, is possible with respect to b and i just in case p is consistent with the conjunction of b and i.

We shall construe Kratzer's propositions as types and we shall take modal bases and ideals to be types as well. To recreate a Kratzerian semantics for necessity and possibility we let the predicates 'nec' and 'poss' have arity ⟨*Type*, *Type*, *Type*⟩.

In order to allow types constructed with these predicates we need to create intensional modal type systems in addition to the intensional type systems we introduced in Chapter 1, p. 32. An intensional modal system of complex types is a family of intensional type systems each of which represents a possibility. Figure 6.2 represents an intensional modal type system where we indicate just the initial three orders of an infinite hierarchy of type orders on just one of the possibilities. Let M be a model, $\langle A, F\rangle$, where A is an assignment to basic types and F an assignment to ptypes as usual. Let $\mathcal{M}$ be an infinite sequence of models, M, indexed by the natural numbers, corresponding to the models for the type systems of each order in an intensional type system. We use $\mathcal{M}_n$ to represent the model for the n-th order in an intensional type system. We use $\mathfrak{M}$ to represent a set of such model sequences, representing the model sequences for each of the possibilities in the intensional modal type system. We characterize an intensional modal system of complex types in (12), repeated in Appendix 10.

(12) An *intensional modal system of complex types based on* $\mathfrak{M}$ is a family, indexed by the natural numbers, of families of quadruples indexed by members of $\mathfrak{M}$:

$$\textbf{TYPE}_{IMC} = \langle \textbf{Type}^n, \textbf{BType}, \langle \textbf{PType}^n, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \mathcal{M}_n\rangle_{\mathcal{M}\in\mathfrak{M}, n\in Nat}$$

where:

1. for each n, $\langle \textbf{Type}^n, \textbf{BType}, \langle \textbf{PType}^n, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \mathcal{M}_n\rangle_{\mathcal{M}\in\mathfrak{M}}$ is a modal system of complex types based on $\{\mathcal{M}_n \mid \mathcal{M} \in \mathfrak{M}\}$
2. for each $\mathcal{M} \in \mathfrak{M}$, $\langle \textbf{Type}^n, \textbf{BType}, \langle \textbf{PType}^n, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \mathcal{M}_n\rangle_{n\in Nat}$ is an intensional system of complex types

[5] Actually, a function which determines a set of propositions for each possible world.
[6] Again relativized to possible worlds.

In terms of our informal proof theoretic notation we can say that if we have an intensional modal type system $\mathbb{G}$, if $\{\Gamma^n\}_{n\in Nat} \in \mathbb{G}$ then $\{\Gamma^n\}_{n\in Nat}$ obeys the rules for intensional systems of types and if for some $m \in Nat$, $\mathcal{G} = \{\Gamma^m \mid \text{for some } X \in \mathbb{G}, \Gamma^m \in X\}$, then $\mathcal{G}$ obeys the rules for modal systems of types.

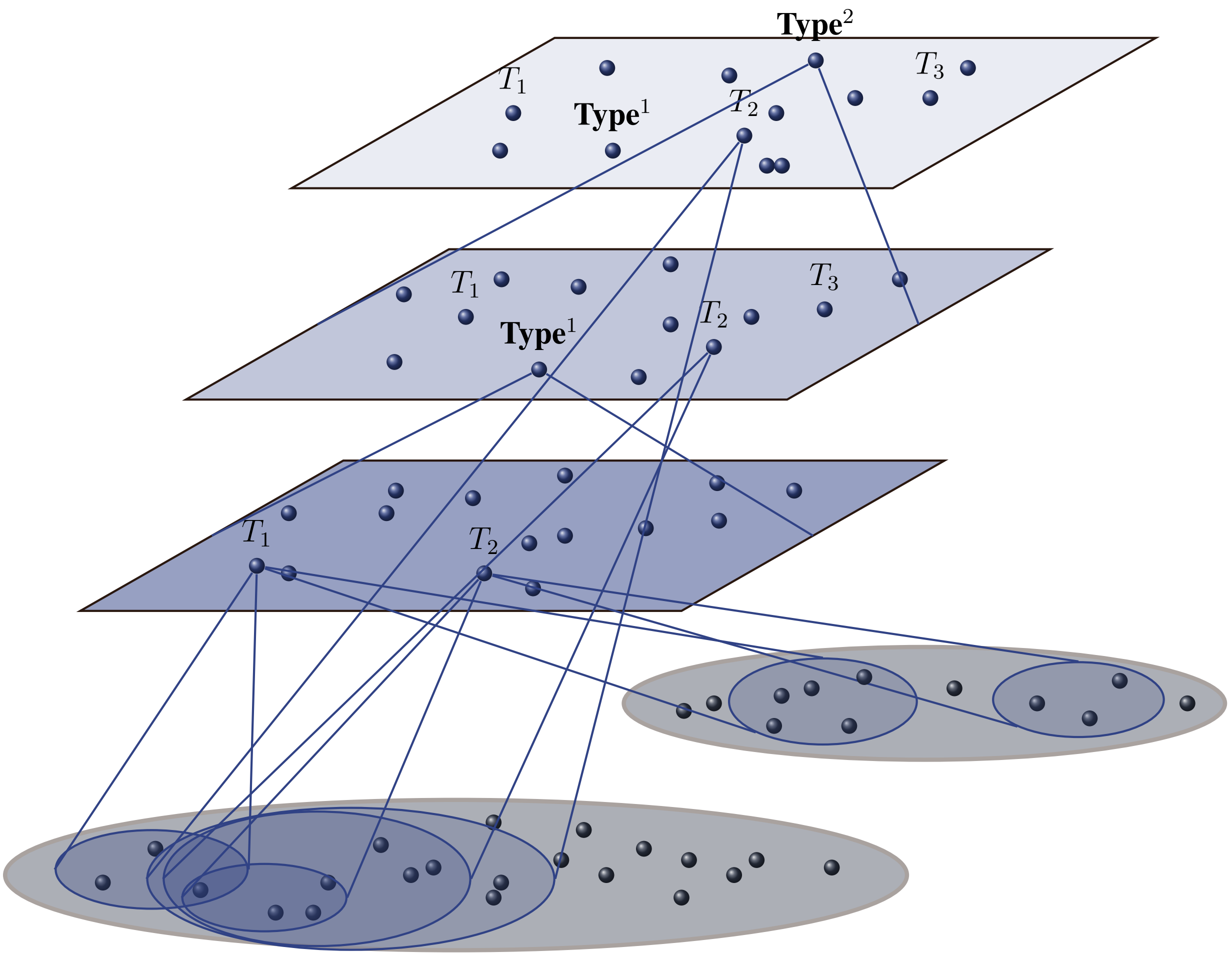


Figure 6.2: Intensional modal type system

Our first suggestion for witness conditions for ptypes constructed with 'nec' are given in (13), assuming that $\mathbb{T}$ is a modal type system, $p \in \mathbb{T}$ and that T, B ("base") and I ("ideal") are types.

(13) Witness condition for 'nec' (version 1)

$s :_p \text{nec}(T, B, I)$ iff $s :_p B$ and for any $p' \in \mathbb{T}$ if for some a, $a :_{p'} B$ (i.e. B is "true") and for some a, $a :_{p'} I$ (i.e. I is "true") then for some a, $a :_{p'} T$ (i.e. T is "true")

A consequence of (13) is that ‘nec(T, B, I)’ is true in some possibility, p. just in case B is true in p and for any possibility, p', if B and I are true in p' then T is true in p'. Building on a basic example from Portner, 2009, p. 49, suppose that T is *Mary-eat-her-broccoli*, B is *Mary-has-broccoli-on-her-plate* and I is *Mary-eats-everything-on-her-plate*. Then according to the definitions in (13) ‘nec(T, B, I)’ is non-empty (i.e. it’s necessary that Mary eat her broccoli) just in case B is true (Mary has broccoli on her plate) and for any of the possibilities we are considering if both B and I are non-empty then T is non-empty, that is, if there’s a situation where Mary has broccoli on her plate and there’s a situation where Mary eats everything on her plate then there’s a situation in which Mary eats her broccoli.

We can treat ‘poss’ in a similar way as in (14).

(14) Witness condition for ‘poss’ (version 1)

$s :_p$ poss(T, B, I) iff $s :_p B$ and for some $p' \in \mathbb{T}$ there is some a, $a :_{p'} B$ (i.e. B is “true”) and there is some a, $a :_{p'} I$ (i.e. I is “true”) and there is some a, $a :_{p'} T$ (i.e. T is “true”)

A consequence of (14) is that poss(T,B,I) is true in some possibility, p, just in case Mary has broccoli on her plate and there is some possibility that we are considering where there’s a situation in which Mary has brocolli on her plate, a situation in which Mary eats everything on her plate and a situation in which Mary eats her broccoli.

The witness conditions (13) and (14) allow different witnesses for the types T, B and I. An alternative is to require the same object to be of these types. This alternative for ‘nec’ is presented in (15).

(15) Witness conditions for ‘nec’ (version 2)

$s :_p$ nec(T, B, I) iff $s :_p B$ and for any $p' \in \mathbb{T}$ and $a :_{p'} (B \wedge I)$, $a :_{p'} T$

For ‘poss’ we have (16).

(16) Witness condition for ‘poss’ (version 2)

$s :_p$ poss(T, B, I) iff $s :_p B$ and for some $p' \in \mathbb{T}$ there is some a, $a :_{p'} ((B \wedge I) \wedge T)$

A disadvantage with (15) is that it does not require the base, B and the ideal, I to be compatible. That is, it allows for ‘nec(T, B, I)’ to be true even if there is no possibility in which there is a witness for $B \wedge I$. This is in contrast to (16) which does require there to be a witness of $B \wedge I$ in some possibility. We could add a compatibility condition to (15) but it suggests that we could formulate a neater definition in terms of relations between the types. We already have a notion of subtype which corresponds to the quantification over p' and a' in (15). Thus we could replace this quantification with $(B \wedge I) \sqsubseteq_{\mathbb{T}} T$. We will use the notation $T_1 \top_{\mathbb{T}} T_2$ to represent that T_1 is compatible with T_2 in the modal system $\mathbb{T}$. This is defined in (17).

(17) T_1 *is compatible with* T_2 *in modal type system* $\mathbb{T}$, $T_1 \top_{\mathbb{T}} T_2$, just in case

there is some $p \in \mathbb{T}$ such that for some a, $a :_p (T_1 \wedge T_2)$

We can now characterize witness conditions associated with 'nec' as (18), as usual assuming some modal system of types, $\mathbb{T}$ and $p \in \mathbb{T}$.

(18) Witness condition for 'nec' (version 3)
$s :_p \text{nec}(T, B, I)$ iff $s :_p B$, $B \top_{\mathbb{T}} I$ and $(B \wedge I) \sqsubseteq_{\mathbb{T}} T$

Similarly, we can define witness conditions for 'poss' as in (19).

(19) Witness condition for 'poss' (version 3)
$s :_p \text{poss}(T, B, I)$ iff $s :_p B$ and $(B \wedge I) \top_{\mathbb{T}} T$

Version 3 of these definitions is interesting because if you have a way of (at least approximately) computing whether one type is a subtype of or compatible with another simply by looking at the types, then you will not have to look at the different possibilities. This makes it possible, for example, to consider all logical possibilities without inspecting all the possibilities but by considering the structure of the types involved. This relates to a more proof theoretic oriented approach to modality. Part of the important insight of Kratzer's approach to modality is that it involves arguments which can be constructed from the modal base and the ideal.

Versions 2 and 3, where we talk of particular situations being witnesses for the types involved rather than just the "truth" of the types, seem to fit better intuitively with the particular broccoli example we are discussing. 'nec(T,B,I)' will be non-empty just in case Mary having broccoli on her plate is compatible with her eating everything on her plate and in all of the possibilities under consideration any situation in which she has broccoli on her plate and eats everything on her plate is also a situation in which she eats her broccoli.

When Kratzer talks of the conversational background consisting of the base and the ideal she often talks about rules that might be encoded there (bodies of laws or regulations in the case of deontic modality). This idea of rules being involved actually fits even better with intuitions about the broccoli example. It is not so much that we are considering possibilities where Mary eats everything on her plate, but rather that we are considering possibilities where there is a rule that Mary eats whatever is on her plate.

It is important for Kratzer that such rules not be logical laws in the sense that they always hold true. For example, a law that cars not park on double yellow lines does not entail that cars do not park on double yellow lines – this is only something that holds true in deontically ideal worlds. This suggests that there could be a role for what Breitholtz (2014a, fthc) calls topoi. A topos in her terms is a dependent type, that is, a function which maps an object of some type to a type. Given a situation of the domain type of the topos, the topos will return a new type. We will introduce a basic type *Topos* which meets the condition in (20).

(20) If τ : *Topos*, then $\tau : \left[\begin{array}{lll} \text{bg} & : & \textit{Type} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{Type}) \end{array}\right]$

We will use $\tau(s)$ to represent $\tau.\text{fg}(s)$. A standard action rule associated with a topos is given in (21).

(21) $$\frac{\tau : \textit{Topos} \qquad \tau \text{ resource}_A \qquad s :_A \tau.\text{bg}}{:_A \tau(s)}$$

That is, if an agent, A, judges a situation s to be of the background (domain) type of a topos, τ, which is available to A as a resource, then A is licensed (afforded) to judge that there is something of type $\tau(s)$.

We will say that topoi associated with this action rule are used *epistemically*. The condition has to do with increasing our knowledge on the basis of a previous judgement. If we judge something, s, to be of the type which is the background of the topos then we can judge that there is something of the type resulting from applying the foreground of the topos to s.

Topoi can also be used *deontically*, that is, they are associated with an action rule which involves creating an event of the type returned by the topos. Such an action rule may represent an affordance as in (22a) or an obligation as in (22b), the latter represented by labelling the action rule with 'oblig'.

(22) a. $$\frac{\tau : \textit{Topos} \qquad \tau \text{ resource}_A \qquad s :_A \tau.\text{bg}}{:_A \tau(s)!}$$

b. $$\frac{\tau : \textit{Topos} \qquad \tau \text{ resource}_A \qquad s :_A \tau.\text{bg}}{:_A \tau(s)!} \text{ oblig}$$

That is, if an agent, A, judges a situation, s, to be of the background type of the topos, then A is allowed/obliged to create (contribute to the creation of) something which is of the type resulting from applying the foreground of the topos to s.

Topoi can be associated with these and other action rules and one topos can be associated with several action rules, that is, the same topos can be used either epistemically or deontically.

We now replace the third "ideal" type argument to the predicates 'nec' and 'poss' with a topos argument, giving them the arity $\langle$*Type*, *Type*, *Topos*$\rangle$. If we need to recreate the option provided by an ideal as a type rather than the topos we can use a topos whose background type is the type *Rec*. That is, it does not place any constraints on the situations in its domain and thus will return a type for any situation. If such a function is a constant function, that is, it returns the same type for any situation, then this will give us the same effect as we obtained when the argument was a type rather than a topos.

We define the witness conditions in (23) for the new version of 'nec', again assuming a modal type system, $\mathbb{T}$ and $p \in \mathbb{T}$.

(23) Witness condition for 'nec' (version 4)

$s :_p \mathrm{nec}(T, B, \tau)$ iff $s :_p B$, $B \sqsubseteq_{\mathbb{T}} \tau.\mathrm{bg}$ and $\tau(s) \sqsubseteq_{\mathbb{T}} T$

In informal terms, (23) says that a situation, s, witnesses that a type, T, is necessary with respect to a background type, B, and a topos, τ just in case s is of the type B, τ is defined on situations of type B and the type resulting from the application of τ to s is such that any situation of that type will be of type T.

Similarly, for 'poss' we have the witness condition in (24).

(24) Witness condition for 'poss' (version 4)

$s :_p \mathrm{poss}(T, B, \tau)$ iff $s :_p B$, $B \sqsubseteq_{\mathbb{T}} \tau.\mathrm{bg}$ and $\tau(s) \top_{\mathbb{T}} T$

(24) says that a situation, s, witnesses that T is possible with respect to B and τ just in case $s : B$, τ is defined on situations of type B and the type resulting from the application of τ to s is consistent with T, i.e. that it is possible for a situation to be of both types.

Let us see how this might play out in our basic example (taken from Portner, 2009, p. 49). Consider (25).

(25) Mary should eat her broccoli

Portner points out that this sentence can receive a bouletic (having to do with desires) intepretation if “we are talking about the fact that Mary loves brocolli” while “if we are trying to enforce the idea that children should eat everything on their plates, it naturally receives a deontic interpretation”. Suppose that b is the brocolli on Mary's plate. For simplicity we will assume b : *Ind*. Let m be Mary and p her plate. Then the type, B, of the base situation could be (26).

$$(26)\quad \begin{bmatrix} \mathrm{x}{=}b & : & \mathit{Ind} \\ \mathrm{c}_1 & : & \mathrm{broccoli(x)} \\ \mathrm{y}{=}m & : & \mathit{Ind} \\ \mathrm{c}_2 & : & \mathrm{child(y)} \\ \mathrm{z}{=}p & : & \mathit{Ind} \\ \mathrm{c}_3 & : & \mathrm{plate(z)} \\ \mathrm{e}_1 & : & \mathrm{have(y,z)} \\ \mathrm{e}_2 & : & \mathrm{on(x,z)} \\ \mathrm{e}_3 & : & \mathrm{love(y,x)} \end{bmatrix}$$

Let us in addition assume that broccoli is food according to the modal type system, $\mathbb{T}$, we are considering, that is, (27) holds.

(27) For any a, brocolli(a) $\sqsubseteq_{\mathbb{T}}$ food(a)

Now let us introduce two topoi, τ_1 and τ_2 represented in (28a and b) respectively.

$$(28)\ \text{a.}\ \tau_1 — \ulcorner\lambda r{:}\begin{bmatrix} \mathrm{x}{:}\mathit{Ind} \\ \mathrm{c}_1{:}\mathrm{food(x)} \\ \mathrm{y}{:}\mathit{Ind} \\ \mathrm{c}_2{:}\mathrm{child(y)} \\ \mathrm{z}{:}\mathit{Ind} \\ \mathrm{c}_3{:}\mathrm{plate(z)} \\ \mathrm{e}_1{:}\mathrm{have(y,z)} \\ \mathrm{e}_2{:}\mathrm{on(x,z)} \end{bmatrix} . \begin{bmatrix} \mathrm{e} & : & \mathrm{eat}(r.\mathrm{y}, r.\mathrm{x}) \end{bmatrix}\urcorner$$

$$\text{b.}\ \tau_2 — \ulcorner\lambda r{:}\begin{bmatrix} \mathrm{x}{:}\mathit{Ind} \\ \mathrm{c}_1{:}\mathrm{food(x)} \\ \mathrm{y}{:}\mathit{Ind} \\ \mathrm{c}_2{:}\mathrm{child(y)} \\ \mathrm{e}_3{:}\mathrm{love(y,x)} \end{bmatrix} . \begin{bmatrix} \mathrm{e} & : & \mathrm{eat}(r.\mathrm{y}, r.\mathrm{x}) \end{bmatrix}\urcorner$$

(28a) associates the type of situation where a child has food on her plate with the type of situation where the child eats that food. This topos is naturally associated with a deontic condition, that is, a child is obliged to create a situation of the type returned by the topos, to eat the food on her plate. (28b) associates the type of situation where there is food which the child loves with the type of situation where the child eats that food. This topos is naturally associated with what we might call a *bouletic* condition, that is, we can use the topos to reason that the child has a desire to create a situation of the type returned by the topos, that is, the child wants to eat the food. This involves a kind of condition which we have not talked about yet which associates types with mental states rather than actions. We will discuss this more in Section 6.4.

The type corresponding to *Mary should eat her broccoli* based on these resources could be either of the types in (29), where T_{broc} is (26) and τ_1 and τ_2 are (28a and b) respectively.

(29) a. $\text{nec}(\left[\text{e:eat}(m, b)\right], T_{\text{broc}}, \tau_1)$
b. $\text{nec}(\left[\text{e:eat}(m, b)\right], T_{\text{broc}}, \tau_2)$

We can now check the witness conditions in (23) against some modal system, $\mathbb{T}$, and possibility, $p \in \mathbb{T}$. Any s which is of the type (29a) has to fulfil the conditions in (30).

(30) a. $s :_p T_{\text{broc}}$
b. $T_{\text{broc}} \sqsubseteq_{\mathbb{T}} \begin{bmatrix} \text{x:}\mathit{Ind} \\ \text{c}_1\text{:food(x)} \\ \text{y:}\mathit{Ind} \\ \text{c}_2\text{:child(y)} \\ \text{z:}\mathit{Ind} \\ \text{c}_3\text{:plate(z)} \\ \text{e}_1\text{:have(y,z)} \\ \text{e}_2\text{:on(x,z)} \end{bmatrix}$
c. $\tau_1(s) \sqsubseteq_{\mathbb{T}} \left[\text{e:eat}(m, b)\right]$

Assuming that s meets (30a), we can check that (30b) holds by noting that anything of the first type will also be of the second type. (In this case, the two types are identical except for (i) 'broccoli' in the first type corresponds to 'food' in the second, but we know from (27) that broccoli is food (ii) the manifest fields in the first type correspond to non-manifest fields in the second, but we know from the definition of singleton types represented by manifest fields that they are subtypes of the corresponding non-singleton type and (iii) the additional field labelled 'e_3', but adding fields to a type creates a subtype of that type.) We can see that (30c) will hold given our characterization of τ_1 in (28) since $\tau_1(s)$ will be (31a) and given that $s : T_{\text{broc}}$, s.y will be m and s.x will be b. Thus $\tau_1(s)$ is identical with (31b).

(31) a. $\left[\ \text{e} \ \ : \ \ \text{eat}(s.\text{y}, s.\text{x})\ \right]$

b. $\left[\begin{array}{lll} \text{e} & : & \text{eat}(m, b) \end{array}\right]$

Thus (30) is checking that the type $\left[\text{e:eat}(m, b)\right]$ is a subtype of itself and, of course, any type is a subtype of itself.

We can make a similar argument for (29b).

Notice that in this worked example we have carefully chosen the labels on our topoi so that they match the types in our example. In a completely explicit treatment we would need to allow for relabelling. We will take up relabelling again in our discussion of intensionality below.

This is an inferential view of modality in the sense that the topoi, which correspond to patterns of inference, have taken over the work of the accessibility relations between possible worlds which Kratzer uses. Note that while it might appear from our formulation of the witness conditions for ‘nec’ and ‘poss’ that we have a definition of modal predicates which does not use the previous notion of modality that we had in terms of possibilities defined in varying the assignments to basic types and ptypes, this is in fact not the case since our definitions of subtyping and compatibility rely on this kind of modality. Thus these definitions have both an inferential flavour (in that they use topoi which are similar to rules of inference) and also a Kripke model flavour in that they use sets of possibilities.

While the use of topoi here gives us something corresponding to accessibility relations in Kratzer’s treatment of modality in Kratzer (1977) (Kratzer, 2012, Chapter 1), it does not yet give us anything corresponding to the notion of ordering source introduced in Kratzer (1981) (Kratzer, 2012, Chapter 2) to deal with the different degrees of modality expressed in examples like

(32) a. Mary absolutely must eat her brocolli
b. Mary must eat her brocolli
c. Mary ought to eat her brocolli
d. Mary should eat her brocolli

While it is not obvious that there is a fixed order of strength in (32) it is nevertheless the case that speakers of English will perceive differences of strength in the modalities having to do with how necessary it is for Mary to eat her brocolli. For that we need the notion of preference structure (Condoravdi and Lauer, 2016) adapted to our TTR approach. We will not undertake this here.

Another important aspect of natural language modality that we will not take up here is its relationship with tense, in particular the future. Clearly if Mary should eat her broccoli, we are thinking of the broccoli as being on her plate before she starts eating it and if she eats all her broccoli then, of course, there will not be broccoli on her plate. The possible eating of the broccoli is temporally after the speech event which is the utterance of *Mary should eat her broccoli*. A number of modal constructions are associated with future tense and we would need to handle this in terms of types of event strings.

While we have no means seen a complete treatment of modality in this section, I hope that this is enough to show that there is a way to explore the treatment of modality which avoids the use of possible worlds and thereby avoids the kinds of problems that possible worlds present for modern treatments of modality.

6.4 Intensionality without possible worlds

In Section 6.1 we discussed problems that have to do with individuating and counting possible worlds. Here we discuss well-known problems that arise when you consider propositions to be the sets of possible worlds[7] which make them true. (For a detailed account of these problems and approaches which have been taken to them in the literature see Égré, 2020.) The central problem is that the sets of possible worlds provide a too coarse-grained analysis of propositions. There are intuitively distinct propositions which are true in the same sets of possible worlds. Standard examples of this are mathematical propositions. Mathematical propositions are not contingent, that is, they are either true in every possible world or false in every possible world. The view of propositions as sets of possible worlds has the consequence that there are only two mathematical propositions: the necessarily true proposition and the necessarily false proposition. It seems unintuitive to reduce a rich field of continuing investigation where new "propositions" are still being discovered and proved or disproved to a field where just two propositions are being discussed. Clearly, mathematics involves a different intuitive notion of proposition that is not modelled by a set of possible worlds. One might be tempted to think that this is a problem about mathematics rather than natural language and that for normal every day dialogue we can ignore this problem. Perhaps we just do not normally talk about necessary propositions or at least what we think of as being necessarily true is in fact relativized in the way that we discussed above in relation to Kratzer's semantics for modality. This is a dangerous route to pursue, not least perhaps because, although many of us do not spend a lot of our time talking about mathematical propositions, we are nevertheless able to express mathematical propositions in natural language and to ignore them would be to rule out something that is part of linguistic activity. There are many of us who are not mathematicians who can nevertheless understand that there is a difference in the content of the two examples in (33).

(33) a. Andrew Wiles proved that two plus two equals four

b. Andrew Wiles proved that Fermat's last theorem is true

[7] Or, if we are concerned with tensed propositions, sets of pairs of possible worlds and times.

If the correct notion of proposition for natural language was that propositions are sets of possible worlds then we should have difficulty in distinguishing the content of these two sentences.

There are non-mathematical candidates for propositions that would be true in all possible worlds. King (2014) points to examples like (34).

(34) a. Bachelors are unmarried
b. Brothers are male siblings

These are examples of what are sometimes called analytic sentences, true in virtue of their meaning. Despite the considerable difficulties with the notion of analyticity (see Rey, 2015, for discussion), it is nevertheless hard to think of a possible world where one of these sentences is true and the other is false. Yet they seem to correspond to different propositions. It does not seem attractive to say that all analytic sentences express the one and only analytic proposition (which in addition is identical with the true mathematical proposition).

There are also examples of sentences, such as those in (35), which we can argue that they express different propositions although they are true in the same possible worlds.

(35) a. Kim sold *Syntactic Structures* to Sam
b. Sam bought *Syntactic Structures* from Kim

An early reference to the equivalence relationship between *buy* and *sell* in the linguistic literature is Fillmore (1970) where it is stated:

> There are no situations that can in themselves be distinguished as buying situations or selling situations; but the choice of one or another of these verbs seems to make it possible to speak of a buying/selling transaction from one of the participant's point of view.

In our terms we would want to say that the ptypes buy(a,b,c) and sell(c,b,a) are distinct types which have the same witnesses. In terms of propositions as sets of possible worlds we would be committed to the claim that these sentences express the same proposition.

The problem is not just a matter of what we intuitively consider to be distinct propositions. It has consequences for the truth of sentences with sentential complements after verbs like *believe* and *know*, the verbs of propositional attitude. If we analyze these verbs in terms of relations between individuals and propositions and we treat propositions as sets of possible worlds then for some individual, a, if a believes/knows p and p is logically equivalent to q (that is, is true in the same possible worlds which in turn means that p and q are the same proposition) then a believes/knows q. This has the unfortunate consequence that once you know one logical truth you know them all. So, for example, somebody who knows that the sum of 2 and 2 is 4 also knows any other mathematical truth (since they are all the same proposition), as well as any analytic truth and any logically valid truths. The problem extends beyond propositions that are true in all, or no, possible worlds. For any two propositions that are true in the same possible worlds (that is, are logically equivalent) if you know or believe one of them then you also know or believe the other. It interacts with the idea (originally advanced by Kripke, 1972) that proper names should be rigid designators, that is, that they should have the same denotation in every possible world. One of the puzzles goes back to discussion by Frege (1892). In the ancient world people believed that the morning star and the evening star were distinct heavenly bodies, whereas they are in fact both the planet Venus. The "morning star" had the name *Phosphorus* and the "evening star" had the name *Hesperus*. If both these names refer to the same planet Venus in all possible worlds then *Phosphorus rose in the morning* expresses the same proposition as *Hesperus rose in the morning*, that is, the two sentences are true in the same possible worlds, though they are not true in all possible worlds. Yet it seems reasonable to say that the Ancients believed that Phosphorus rose in the morning but that they did not believe that Hesperus rose in the morning. Frege's original puzzle, which is also problematic for the view that propositions are sets of possible worlds concerned the difference between *The Ancients believed that Hesperus is Hesperus* (true if they believed in the law of self identity which they presumably did) and *The Ancients believed that Hesperus is Phosphorus* (false, since it was an astronomical discovery that both Hesperus and Phosphorus are the planet Venus). Yet both *Hesperus is Hesperus* and *Hesperus is Phosphorus* represent the same proposition, the one that is true in all possible worlds. As we noted in Chapter 4 this problem is related to Kripke's Paderewski puzzle which we discussed there and we will build on our analysis of proper names in that chapter in our analysis of the attitudes in this chapter.

The example of the equivalence of *buy* and *sell* may initially seem like an argument for the straightforward possible worlds approach when we consider propositional attitudes like *believe* and *know*. It seems impossible that any rational agent who believes or knows one of (35) would not know or believe the other. However, there are other attitude predicates where it does seem feasible to make the distinction. The sentences in (36) do not seem to be contradictory.

(36) a. Chris was happy that Kim bought *Syntactic Structures* from Sam

b. Chris was not happy that Sam sold *Syntactic Structures* to Kim

There are other non-attitude predicates which also make the distinction. For example, in Sweden it is illegal to buy sex but not illegal to sell sex which has important consequences for who gets punished in a situation where sex is bought and sold. Thus the sentences in (37) are consistent when considering Swedish law.

(37) a. It was illegal that Kim bought sex from Sam

b. It was not illegal that Sam sold sex to Kim

These problems have been well known since the early days of formal semantics. There is an excellent overview of the discussion up to the end of 1970's in Dowty *et al.* (1981), 170ff. An up to date discussion from a more philosophical perspective is given by Égré (2020). Partee (1979) provides an important account of relevant issues from a linguistic perspective. For a modern update of Partee's view see Partee (2014). For some modern philosophical views of propositions which go in somewhat similar directions to the proposals here, linking propositions to perception and action, see King *et al.* (2014).

Our basic strategy here is to replace the notion of propositions as sets of possible worlds with the notion of propositions as types, which goes back to work in intuitionistic logic (see discussion by Ranta, 1994, for a relation of this idea to linguistic semantics, and Wadler, 2015, for an overview of the history of the idea from the perspective of logic and computer science). There is a more sophisticated view of propositions in TTR which was advanced by Ginzburg (2012) and used, for example, in Cooper *et al.* (2015). This is that we should regard propositions as pairs of a situation and a type (that is, a record with two fields). This is the notion of Austinian proposition which goes back to Barwise and Perry (1983) who coined the term because of the proposal in Austin (1961) that propositions should incorporate the part of the world which they are true (or false) of. Both of these notions of proposition exploit the intensionality of types, the fact that you can have two distinct types with the same set of witnesses. A type used as a proposition is true just in case there is something of the type. This makes types as propositions parallel to what was called a Russellian proposition in Barwise (1989), Chap. 11. An Austinian proposition is true just in case the situation in the proposition is of the type of the proposition. An Austinian proposition is a way of reifying a judgement, that is, it gives us an object in our type theoretic universe which corresponds to the act of judging a particular situation to be of a type (a record of such a judgement). This means that if a Russellian proposition is true then there is an Austinian proposition containing the same type which is true. If an Austinian proposition is true then the corresponding Russellian proposition is true. If a Russellian proposition is false then any Austinian proposition containing the same type is also false. However, if an Austinian proposition is false, then we cannot conclude from this either the truth or falsity of the corresponding Russellian proposition. We know that the particular situation in the Austinian proposition is not of the type in the Austinian proposition but this tells us nothing about whether there is some other situation of the type.

Neither "proposition" nor "Russellian proposition" are technical terms in TTR. This is because we can judge any type to be non-empty ("true") or empty ("false") and thus any type can be used as a proposition. In practice, however, we will take record types (intuitively, types of situations) to be what corresponds to the intuitive notion of propositions that can be expressed in natural language. The simplest theory of verbs of propositional attitude like *believe* and *know* on this kind of view would be that they correspond to predicates which express relations between individuals and record types, that is, there are predicates 'believe' and 'know' with arity $\langle$*Ind*,*RecType*$\rangle$. This means that we will have a ptype like (38) where a is an individual and T is a record type.

(38) believe(a, T)

What does it mean for this type to be non-empty? We will say that it involves finding a match, in the sense introduced in Chapter 4, for T in a's long term memory. In the terms introduced in Chapter 4 this means that if r is a's total information state, then a's long term memory will be r.ltm, which is a record type, a type representing how the world would be if a's long term memory were true. Thus we are matching the type T, a record type which is the second argument of 'believe', against another record type corresponding to a's long term memory. Note that according to the proposal for matching in Chapter 4 this involves finding a relabelling for T. The match obtains if there is a relabelling, η, of T, such that r.ltm$\sqsubseteq$ $[T]_\eta$, where $[T]_\eta$ is the result of relabelling T by η (see Chapter 4, p. 177). Let us introduce an abbreviatory notation for this as in (39).

(39) $T_1 \sqsubseteq_{\leadsto} T_2$ just in case there is some relabelling, η, of T_2 such that $T_1 \sqsubseteq [T_2]_\eta$.

Our preliminary witness conditions for believe(a, T) are given in (40). (We will modify this below.)

(40) e : believe(a, T) iff
e : ltm(a, T')
and $T' \sqsubseteq_{\leadsto} T$

The fact that relabelling is involved in the matching process is important for the analysis of belief because it means that (41) holds.

(41) If s : believe(a, T), then for any relabelling, η, of T, s : believe(a, $[T]_\eta$)

This means that the choice of particular labels in a record type is not relevant when we compute whether an agent stands in the belief (or other attitude) relation to a record type. Note also that, given the way we have defined relabelling in terms of paths, that record types which are structured differently, as in (42a,b), will also count as relabellings of each other, in this example in virtue of the relabelling (42c).

(42) a. $\begin{bmatrix} \ell_1 & : & \begin{bmatrix} \ell_2 & : & T_1 \\ \ell_3 & : & T_2 \end{bmatrix} \\ \ell_4 & : & T_3 \end{bmatrix}$

b. $\begin{bmatrix} \ell_1 & : & T_1 \\ \ell_2 & : & \begin{bmatrix} \ell_3 & : & T_2 \\ \ell_4 & : & T_3 \end{bmatrix} \end{bmatrix}$

c. $\ell_1.\ell_2 \rightsquigarrow \ell_1$
$\ell_1.\ell_3 \rightsquigarrow \ell_2.\ell_3$
$\ell_4 \rightsquigarrow \ell_2.\ell_4$

Thus any agent who stands in the belief-relation to (42a) will also stand in the belief-relation to (42b) and *vice versa*. The intuition here is that two agents will have the same beliefs even though they structure the information differently in their separate long term memories.

This can be contrasted with proposals for structured meanings in the possible worlds literature, starting with Lewis (1972), who based his idea on the notion of intensional isomorphism from Carnap (1956), and developed by Cresswell (1985). The idea here is that you alleviate the coarse-grainedness of the possible worlds analysis of propositions by keeping around the functions and arguments that are used to compute the set of possible worlds corresponding to a sentence during its derivation. (A computer scientist could usefully compare this notion of structured meaning to *lazy evaluation*, discussed in relation to computational semantics by van Eijck and Unger, 2010.) The structured meaning is then a semantic derivation structure which is used to calculate synonymy and as the second argument of predicates like *believe*. One problem with this approach is that sentences with radically different structure which nevertheless intuitively express the same proposition may correspond to different structured meanings. One possible example is the active and passive sentences in (43).

(43) a. Kim sold the book to Sam
b. Sam was sold the book by Kim

It is hard to think of a way in which a competent native speaker of English could believe one of these but not the other. Such examples depend very much on the way in which you analyze them and how you set up the relation between syntax and semantics. For example, if you believe that compositional semantics is not defined directly on English syntax but on a logical form derived from English syntax and you are careful to relate both sentences to the same logical form, then, of course, both sentences could be related to the same structured meaning. Another kind of example which is possibly more difficult to handle with such machinery is cases of speakers of different languages with radically different structure who nevertheless intuitively share the same belief.

This kind of theory when viewed from the perspective of the theory presented in this book presents a rather odd view of the phenomena. It first proposes a theory of propositions which is obviously too coarse-grained to model the propositional attitudes. It then tries to fix this by using the derivational structure involved in reading these propositions off the syntax of the natural language. When this turns out to be too fine-grained a wholly new representation, logical form, is introduced to fix this new problem. The status of logical form is in our terms mysterious. It is neither based on the utterance situation nor on the situation types used to construct the content associated with the utterance situation. It is an additional language introduced in order to fix problems involved in interpreting utterances directly, a language which mediates between the utterance and the content. If logical form is more amenable to semantic interpretation than natural language one might raise the question why we do not speak in logical forms rather than the way we do. It is hard to imagine what the realistic status of this intermediate language should be either in terms of the utterance situation, the type of situation associated with the content or neurological events associated with perceiving or conceiving either of these.

A second problem for the structured meaning approach is that it tells us nothing about cases where no syntactic structure is involved, for example, proper names which have the same referent like *Hesperus* and *Phosphorus* or synonymous words like *groundhog* and *woodchuck*. This is pointed out by Dowty *et al.* (1981) and also by Égré (2020) who attributes the first mention of it to Mates (1952).

The fact that matching is involved in the logic of belief rules out two important ways (relating to labelling and the internal structure of record types) in which record types could be too fine-grained to give an analysis of intuitive propositions. In general it seems preferable to start from objects that are too fine-grained since we can then set about finding ways of collapsing distinctions rather than starting out with something (like sets of possible worlds) which are not fine-grained enough and trying to add things to it to make the finer distinctions.

Another advantage of this strategy is that it offers possibilities for varying the fineness of the grain for different cases. Thus while we can understand that (37) can be consistent, it is much harder to think that both of (44a,b) could be true.

(44) a. Chris believes that Kim bought sex from Sam

b. Chris does not believe that Sam sold sex to Kim

The best we can do to make sense of (44) as a pair of consistent sentences is that Chris is either irrational in her beliefs or does not have sufficient understanding of the language, or that somehow the equivalence between *buy* and *sell* has been suspended. This seems very different from (37).

In the case of *believe* we have suggested that the type represented by the complement has to be matched against the long-term memory of the believer in order for the sentence to be true. The kind of matching introduced in Chapter 4 involves not only relabelling but also subtyping. Suppose that Chris's long term memory is modelled by the type (45a) and that the content of an utterance of *Sam sold sex to Kim* is the type (45b).

(45) a. $\left[\begin{array}{lll} \vdots & & \\ \text{id}_i & : & \left[\begin{array}{lll} \text{e} & : & \text{buy(kim, sex, sam)} \end{array}\right] \\ \vdots & & \end{array}\right]$

b. $\left[\begin{array}{lll} \text{e} & : & \text{sell(sam, sex, kim)} \end{array}\right]$

Is there a match for (45b) in (45a)? The answer is "yes". The relevant relabelling is (46a) and the result of applying that relabelling to (45b) is (46b).

(46) a. $\text{e} \rightsquigarrow \text{id}_i.\text{e}$

b. $\left[\begin{array}{lll} \text{id}_i & : & \left[\begin{array}{lll} \text{e} & : & \text{sell(sam, sex, kim)} \end{array}\right] \end{array}\right]$

We can see that (45a) is a subtype of (46b) in virtue of the fact in (47) — any event of buying is also an event of selling.

(47) $\text{buy(kim, sex, sam)} \sqsubseteq \text{sell(sam, sex, kim)}$

In this way we can obtain the correct level of granularity for *believe*. Consider now (48a) where we have the verb *say* instead of *believe* and a situation where the actual utterance that Chris made was (48b).

(48) a. Chris said that Sam sold sex to Kim

b. Kim bought sex from Sam

Is (48a) true in this case? It seems that we answer this question differently depending on how close the match between the reported speech and the original utterance has to be for the purposes at hand. Ginzburg and Cooper (2014) treat direct quotation in terms of a similarity metric on types which is associated with the context. In different contexts we require different similarity metrics. In some contexts (49) might be considered close enough given that what Chris had said originally was (48b).

(49) Chris said, "Sam sold sex to Kim."

This might be especially be true if Chris's original utterance was in a language other than English. Here I would like to say that indirect speech cases like (48) also involve a similarity metric given by the context and that similarity metrics associated with indirect speech in general can be looser that those associated with direct speech where we often look not only at the content of the original utterance but also its exact form of words. So according to some similarity metrics (48a) will be true and for others it will be false. It will be true intuitively if the content of its complement is close enough for current purposes to the content of Chris's original utterance.

We can assimilate our treatment of belief to this general treatment involving similarity metrics by defining a similarity metric that says that the type representing an agent's long term memory is similar to the type which is the content of the belief complement if the complement content matches the long term memory type in the way we have described. We will argue below that there is an advantage to making this assimilation since the criteria we use for whether an agent has a certain belief seem to vary according to the purposes we have at hand in the current context.

One of the distinctions that it seems to be possible to make in similarity metrics involves different kinds of subtyping. We have defined subtyping so that for two types, T_1 and T_2, T_1 is a subtype of T_2 just in case for any a, if $a : T_1$ then $a : T_2$ and that this holds no matter what assignment is made to basic types and ptypes. Now consider the two examples of subtyping in (50).

(50) a. $\begin{bmatrix} \ell_1 & : & T_1 \\ \ell_2 & : & T_2 \end{bmatrix} \sqsubseteq \begin{bmatrix} \ell_1 & : & T_1 \end{bmatrix}$

b. sell(a, b, c) $\sqsubseteq$ buy(c, b, a)

(50a) holds because of the general characterization of our type theory. It is, if you like, "hard-wired" into the type theoretic system. There is no way that you could construct a type system of the kind TTR characterizes which does not require (50a). (50b), on the other hand, holds only in virtue of a "postulate" that we have added to the general system relating to the particular predicates 'buy' and 'sell'. Just as Montague (1973) introduced what have come to be known as meaning postulates in his system as "restrict[ing] attention to those interpretations of intensional logic in which the following formulas are true", a postulate concerning the equivalence of selling and buying events in TTR means that we are restricting attention to possibilities (assignments to basic types and ptypes) in which the equivalence holds. According to the general definitions of TTR (not including such postulates) it is possible to construct a system where the equivalence does not hold. We will refer to (50a) as an instance of *structural subtyping* and (50b) as an instance of *postulated subtyping*. It appears that natural languages can distinguish between these different kinds of subtyping in the kind of matching that is required by predicates which take types as arguments. In the case of believe(a, T) we say that this is instantiated (non-empty) just in case a's long term memory is characterized by a type which, modulo relabelling, is a subtype (either structural or postulated) of T. On the other hand, if we think of a set of laws as characterizing, among other things, a set of forbidden types of situations, then illegal(T) would be instantiated just in case T is, modulo relabelling, a structural subtype of one of the forbidden types.

The distinction between structural and postulated subtyping also gives us a clue on how to deal with groundhogs and woodchucks. Structural subtyping is hardwired into the system. Any cognitive system which implements types will also have structural subtyping, assuming TTR is the right type theory for cognitive systems. Any such system will also have the capability to include postulated subtyping. But exactly which postulates the system has is a matter of learning. Different agents will acquire different postulates depending on their experience. While it is hard to imagine a competent speaker of English not knowing the equivalence between buying and selling it is very easy to suppose that a competent speaker does not know the equivalence between woodchucks and groundhogs. Indeed it would be natural for speakers to assume that the words *woodchuck* and *groundhog* are associated with distinct types and an agent would need some kind of evidence to establish an equivalence between the types. It would be possible for an agent who has not acquired the postulates that establish the equivalence to believe that a woodchuck is in the garden but not to believe that a groundhog is in the garden. However, an agent who has acquired the equivalence would have to believe or disbelieve both. Thus the claim in (51) seems contradictory.

(51) Kim knows that woodchucks are the same as groundhogs and believes that a woodchuck is in the garden but does not believe that a groundhog is in the garden

The only way we can make sense of Kim believing something about a woodchuck but not about a groundhog is that Kim is unaware that woodchucks and groundhogs are the same animal. Thus getting the semantics of these attitude reports right is not simply a matter of having a finegrained enough semantics to distinguish between *woodchuck* and *groundhog* but also in linking this finegrainedness to a lack of knowledge about equivalence on the agent's part.

Suppose that Kim believes a woodchuck is in the garden and does not have the postulated equivalence between woodchuck and groundhog. It would seem from what we have said above that it does not follow that Kim believes that a groundhog is in the garden, and indeed there is a sense in which this is right, if we are taking account of subtyping according to Kim's postulates. Suppose, however, that I do know that woodchucks and groundhogs are the same animal. It seems that I can truthfully report that Kim believes that a groundhog is in the garden, using my knowledge that woodchucks and groundhogs are the same, even though Kim would not herself necessarily assent to a claim: "There's a groundhog in the garden". There is a systematic ambiguity in reports of this kind as to whether the match with Kim's long term memory is computed using the postulates available in Kim's resources or the postulates available in the reporter's resources. Most of the time we do not notice this distinction because it only arises in the case where there is this particular discrepancy between the resources available to the two agents. But it is important to note that in this case there is no one answer to the question *Does Kim believe that a groundhog is in the garden?*. In one sense she does not, and in another sense she does. On the reading where the reporter uses her own postulates it seems that there is a relationship with quotation in translation. Suppose that Kim is a monolingual speaker of German and has a belief which would be reported in German as "Ein Waldmurmeltier ist im Garten". The way in which this belief should be reported in English has to depend entirely on the reporter's resources concerning the correspondences between the contents of *Waldmurmeltier*, *groundhog* and *woodchuck*.

There is a similar systematic ambiguity to that we saw with reporting beliefs about woodchucks and groundhogs in our reporting of ancient beliefs about Hesperus and Phosphorus. Did the ancients believe that Venus rose in the morning? In one sense they did not, since they did not know that the heavenly body which they called Hesperus was in fact Venus. In another sense they did, since the heavenly body which they called Hesperus is in fact (according to the reporter's resources) Venus. The change in long term memory of an ancient who learns that Hesperus and Phosphorus are identical is parallel to that discussed in relation to example (75) and subsequent examples in Chapter 4 except that two proper names are involved rather than one. The type of the ancients' long term memory in their state of ignorance could be a subtype of (52) for some natural numbers i, j, k and l.

(52)
$$\begin{bmatrix} \text{id}_i : \begin{bmatrix} \text{x}:\textit{Ind} \\ \text{e}:\text{named(x, “Hesperus”)} \end{bmatrix} \\ \text{id}_j : \begin{bmatrix} \text{e}:\text{rise\_in\_the\_evening}(\Uparrow\text{id}_i.\text{x}) \end{bmatrix} \\ \text{id}_k : \begin{bmatrix} \text{x}:\textit{Ind} \\ \text{e}:\text{named(x, “Phosphorus”)} \end{bmatrix} \\ \text{id}_l : \begin{bmatrix} \text{e}:\text{rise\_in\_the\_morning}(\Uparrow\text{id}_k.\text{x}) \end{bmatrix} \end{bmatrix}$$

Upon the ancients' learning that Hesperus and Phosphorus are the same object, (52) would be updated to (53a) which is identical with (53b).

(53) a.
$$\begin{bmatrix} \text{id}_i : \begin{bmatrix} \text{x}:\textit{Ind} \\ \text{e}:\text{named(x, “Hesperus”)} \end{bmatrix} \\ \text{id}_j : \begin{bmatrix} \text{e}:\text{rise\_in\_the\_evening}(\Uparrow\text{id}_i.\text{x}) \end{bmatrix} \\ \text{id}_k : \begin{bmatrix} \text{x}:\textit{Ind} \\ \text{e}:\text{named(x, “Phosphorus”)} \end{bmatrix} \\ \text{id}_l : \begin{bmatrix} \text{e}:\text{rise\_in\_the\_morning}(\Uparrow\text{id}_k.\text{x}) \end{bmatrix} \end{bmatrix} \mathrel{\dot{\wedge}} \begin{bmatrix} \text{id}_i : \begin{bmatrix} \text{x}:\textit{Ind} \end{bmatrix} \\ \text{id}_k : \begin{bmatrix} \text{x}{=}\Uparrow\text{id}_i.\text{x}:\textit{Ind} \end{bmatrix} \end{bmatrix}$$

b.
$$\begin{bmatrix} \text{id}_i : \begin{bmatrix} \text{x}:\textit{Ind} \\ \text{e}:\text{named(x, “Hesperus”)} \end{bmatrix} \\ \text{id}_j : \begin{bmatrix} \text{e}:\text{rise\_in\_the\_evening}(\Uparrow\text{id}_i.\text{x}) \end{bmatrix} \\ \text{id}_k : \begin{bmatrix} \text{x}{=}\Uparrow\text{id}_i.\text{x}:\textit{Ind} \\ \text{e}:\text{named(x, “Phosphorus”)} \end{bmatrix} \\ \text{id}_l : \begin{bmatrix} \text{e}:\text{rise\_in\_the\_morning}(\Uparrow\text{id}_k.\text{x}) \end{bmatrix} \end{bmatrix}$$

Note that (53) could be construed as corresponding to a state of mind where an ancient would still refer to Venus as "Hesperus" in connection with evening rising events and "Phosphorus" in connection with morning rising events even though she was aware that they were the same object. The structure of the memory associates the different name with certain types of events. This seems intuitively correct.

Recall that 'SemPropName', which characterizes the meanings of proper names, is defined as in (54).

(54) If T_{phon} is a phonological type, then SemPropName(T_{phon}) is

$$\ulcorner\lambda c{:}\ \begin{bmatrix} \textit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix} \text{x}{:}\textit{Ind} \\ \text{e}{:}\text{named}(\text{x},\, T_{\text{phon}}) \end{bmatrix} \end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(c.\mathfrak{c})\urcorner$$

According to the account of proper names we gave in Chapter 4, the background type (that is, the domain type of the function, specifying the type of the context) has to be matched against the gameboard or failing that against the long-term memory or failing that added to the gameboard, before the new information can be integrated. The assumption in that discussion was that the relevant long-term memory was that of the agent integrating the utterance. Now we are raising the issue of whose long-term memory is the relevant one to check. There are three long-term memories which can be relevant in a belief report: that of the agent integrating the utterance of the report (that is, the same long-term memory as we were considering in Chapter 4), the long-term memory of the reporter and the long-term memory of the subject of the report (the "believer"). Obviously it is the information state of the agent integrating the report that we are primarily concerned with as it is this integration process which we are trying to explain. This agent does not, of course, have direct access to the long-term memories of either the reporter or the subject of the report. (The integrator's brain is not wired to either the reporter's or the subject's brain.) However, the integrator can form views of the nature of their long-term memories using as evidence, among other things, utterances made by them or utterances made my others about them. Such information about the long-term memories of the reporter and subject can be incorporated in the integrator's long-term memory. That is, among the beliefs we have encoded in long-term memory we have beliefs concerning what others believe. Consider the type characterizing long-term memory in (55).

(55)
$$\left[\begin{array}{l}
\text{id}_1:\begin{bmatrix}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Venus”)}\end{bmatrix}\\
\text{id}_2:\begin{bmatrix}\text{e}:\text{rise_in_the_evening}(\Uparrow\text{id}_1.\text{x})\end{bmatrix}\\
\text{id}_3:\begin{bmatrix}\text{e}:\text{rise_in_the_morning}(\Uparrow\text{id}_1.\text{x})\end{bmatrix}\\
\text{id}_4:\begin{bmatrix}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Homer”)}\end{bmatrix}\\
\text{id}_5:\left[\begin{array}{l}
\text{id}_1{=}\left[\begin{array}{l}
\text{id}_1:\begin{bmatrix}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Hesperus”)}\end{bmatrix}\\
\text{id}_2:\begin{bmatrix}\text{e}:\text{rise_in_the_evening}(\Uparrow\text{id}_1.\text{x})\end{bmatrix}\\
\text{id}_3:\begin{bmatrix}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Phosphorus”)}\end{bmatrix}\\
\text{id}_4:\begin{bmatrix}\text{e}:\text{rise_in_the_morning}(\Uparrow\text{id}_3.\text{x})\end{bmatrix}
\end{array}\right]:\mathit{RecType}\\
\text{id}_2:\begin{bmatrix}\text{e}:\text{believe}(\Uparrow^2\text{id}_4.\text{x},\ \Uparrow\text{id}_1)\end{bmatrix}\\
\text{id}_3{=}\left[\begin{array}{l}
\text{id}_1:\begin{bmatrix}\text{x}{=}\Uparrow^2\text{id}_1.\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Venus”)}\end{bmatrix}\\
\text{id}_3:\begin{bmatrix}\text{x}{=}\Uparrow^2\text{id}_1.\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Venus”)}\end{bmatrix}
\end{array}\right]:\mathit{RecType}\\
\text{id}_4:\text{pov}(\text{id}_3,\ \text{id}_1)
\end{array}\right]
\end{array}\right]$$

Here the type in the ‘$\text{id}_5.\text{id}_3$’-field in (55) is a *point of view*[8]on the type in the ‘$\text{id}_5.\text{id}_1$’-field. A point of view on a type, T, is a type which has labels which overlap with those of T and represents an alternative take on the fields with corresponding labels. In (55), we introduce a predicate ‘pov’ with arity $\langle$*RecType*, *RecType*$\rangle$ such that pov(T_1,T_2) will have a witness just in case T_1 is a point of view on T_2. Here what is represented is that both what Homer calls Hesperus and what Homer calls Phosphorus is what the agent whose long term memory is represented in (55) would call Venus. We can obtain a complete alternative version of the original type by taking the asymmetric merge (see Appendix 12) of the original type with the point of view. Thus in this case we can obtain (56a) which is identical with (56b).

(56) a.
$$\left[\begin{array}{l}
\text{id}_1:\begin{bmatrix}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Hesperus”)}\end{bmatrix}\\
\text{id}_2:\begin{bmatrix}\text{e}:\text{rise_in_the_evening}(\Uparrow\text{id}_1.\text{x})\end{bmatrix}\\
\text{id}_3:\begin{bmatrix}\text{x}:\mathit{Ind}\\ \text{e}:\text{named(x, “Phosphorus”)}\end{bmatrix}\\
\text{id}_4:\begin{bmatrix}\text{e}:\text{rise_in_the_morning}(\Uparrow\text{id}_3.\text{x})\end{bmatrix}
\end{array}\right]\ \boxed{\underset{\cdot}{\wedge}}$$

[8]Notionally our notion of point of view here is related to the notion of perspective discussed by Asudeh and Giorgolo (2016). An important difference, however, is that points of view are structured types whereas perspectives are similar to Montague’s possible worlds in being atomic elements with no internal structure which can be used to explain how they relate to each other.

$$\left[\begin{array}{ll}\text{id}_1: & \left[\begin{array}{l}\text{x}{=}{\Uparrow}^2\text{id}_1.\text{x}{:}\textit{Ind}\\ \text{e:named(x, “Venus”)}\end{array}\right]\\ \text{id}_3: & \left[\begin{array}{l}\text{x}{=}{\Uparrow}^2\text{id}_1.\text{x}{:}\textit{Ind}\\ \text{e:named(x, “Venus”)}\end{array}\right]\end{array}\right]$$

b. $$\left[\begin{array}{ll}\text{id}_1: & \left[\begin{array}{l}\text{x}{=}{\Uparrow}^2\text{id}_1.\text{x}{:}\textit{Ind}\\ \text{e:named(x, “Venus”)}\end{array}\right]\\ \text{id}_2: & \left[\text{e:rise\_in\_the\_evening}({\Uparrow}\text{id}_1.\text{x})\right]\\ \text{id}_3: & \left[\begin{array}{l}\text{x}{=}{\Uparrow}^2\text{id}_1.\text{x}{:}\textit{Ind}\\ \text{e:named(x, “Venus”)}\end{array}\right]\\ \text{id}_4: & \left[\text{e:rise\_in\_the\_morning}({\Uparrow}\text{id}_3.\text{x})\right]\end{array}\right]$$

In order to account for belief when a point of view is involved we need to revise the witness conditions for ‘believe’ which were given in (40). In order to state this revision it will be useful to have a notation for saying that an object, a, is of a type, T, or one of a’s components is of type T. We shall use $a :_{\underline{\varepsilon}} T$ to represent this. The characterization of this notation is given in (57).

(57) $a :_{\underline{\varepsilon}} T$ iff either $a : T$ or for some $b\ \varepsilon\ a$, $b : T$

The revision for the witness condition associated with ‘believe’ is given in (58) and involves replacing the orginal biconditional with a conditional and adding an additional conditional to cover the case for a point of view:

(58) e : believe(a, T) if
  e : ltm(a, T')
  and $T' \sqsubseteq_{\rightsquigarrow} T$

e : believe(a, T) if
  $e :_{\underline{\varepsilon}}$ believe(a, T_1)
  $e :_{\underline{\varepsilon}}$ pov(T_2, T_1)
  and $T_1 \boxed{\dot{\wedge}} T_2 \sqsubseteq_{\rightsquigarrow} T$

Suppose, contrary to fact, that Homer encountered Pythagoras, who believed that the morning star and the evening star were identical and Homer, while perfectly aware of Pythagoras’ belief, maintained a distinction between the two objects and that Pythagoras used the name “Venus”[9] to refer to both Hesperus and Phosphorus. A type of Homer’s long-term memory could be (59).

[9] Or more correctly from the historical point of view: “Aphrodite”.

(59)
$$\left[\begin{array}{ll}
\text{id}_1: & \left[\begin{array}{l} \text{x}:\textit{Ind} \\ \text{e:named(x, “Hesperus”)} \end{array}\right] \\
\text{id}_2: & \left[\text{e:rise_in_the_evening}(\Uparrow\text{id}_1.\text{x})\right] \\
\text{id}_3: & \left[\begin{array}{l} \text{x}:\textit{Ind} \\ \text{e:named(x, “Phosphorus”)} \end{array}\right] \\
\text{id}_4: & \left[\text{e:rise_in_the_morning}(\Uparrow\text{id}_3.\text{x})\right] \\
\text{id}_5: & \left[\begin{array}{l} \text{x}:\textit{Ind} \\ \text{e:named(x, “Pythagoras”)} \end{array}\right] \\
\text{id}_6: & \left[\begin{array}{ll}
\text{id}_1= & \left[\begin{array}{ll}
\text{id}_1: & \left[\begin{array}{l} \text{x}:\textit{Ind} \\ \text{e:named(x, “Venus”)} \end{array}\right] \\
\text{id}_2: & \left[\text{e:rise_in_the_morning}(\Uparrow\text{id}_1.\text{x})\right] \\
\text{id}_3: & \left[\text{e:rise_in_the_evening}(\Uparrow\text{id}_1.\text{x})\right]
\end{array}\right]:\textit{RecType} \\
\text{id}_2: & \left[\text{e:believe}(\Uparrow^2\text{id}_5.\text{x},\ \Uparrow\text{id}_1)\right] \\
\text{id}_3= & \left[\text{id}_1:\left[\text{x}{=}\Uparrow^3\text{id}_1.\text{x},\Uparrow^3\text{id}_3.\text{x}:\textit{Ind}\right]\right]:\textit{RecType} \\
\text{id}_4: & \text{pov}(\text{id}_3,\ \text{id}_1)
\end{array}\right]
\end{array}\right]$$

Note that here we have generalized the convenient notation for manifest fields. The manifest field in the type (under ‘$\text{id}_6.\text{id}_3$’) $\left[\text{x}{=}\Uparrow^3\text{id}_1.\text{x},\Uparrow^3\text{id}_3.\text{x}:\textit{Ind}\right]$ requires that the value in the ‘x’-field is identical with the value of the two values in the fields at the top level on the paths ‘$\text{id}_1.\text{x}$’ and ‘$\text{id}_3.\text{x}$’. We use the notation $\left[\ell{=}a, b, \ldots{:}T\right]$ to represent $\left[\ell{:}T_a \wedge T_b \wedge \ldots\right]$.

A more complex point of view in place of the type given under ‘$\text{id}_6.\text{id}_3$’ is (60).

(60)
$$\left[\begin{array}{ll}
\text{id}_1: & \left[\text{x}{=}\Uparrow\text{id}_4.\text{x},\Uparrow\text{id}_5.\text{x}:\textit{Ind}\right] \\
\text{id}_4: & \left[\begin{array}{l} \text{x}{=}\Uparrow^3\text{id}_1.\text{x}:\textit{Ind} \\ \text{e:named(x, “Hesperus”)} \end{array}\right] \\
\text{id}_5: & \left[\begin{array}{l} \text{x}{=}\Uparrow^3\text{id}_3.\text{x}:\textit{Ind} \\ \text{e:named(x, “Phosphorus”)} \end{array}\right]
\end{array}\right]$$

With (60) substituted for the record type in ‘$\text{id}_6.\text{id}_3$’ in (59), Homer might now truthfully report (61).

(61) Pythagoras believes that Hesperus rises in the evening and Phosphorus rises in the morning (though, of course, he believes they are both something called Venus)

In case (61) does not seem convincing, let us consider a more modern story (given in (62)) where there actually are two individuals who are mistakenly considered to be one individual.

(62) Tom and Bill Smith are identical twins who are both employed as teachers at the same school (a source of endless confusion for staff and students alike). Sam is a new girl spending her first day at the school. Early in the morning Tom, for whom Sam has the name “Mr Smith”, tells her class, “There will be Geometry at 11”. Sam thinks he said ‘There will be Geography at 11’. Later in the morning Bill addresses her class and Sam thinks he is the same “Mr Smith” she saw earlier. Bill says, “There will be French at 11:30.” Sam, who had been too nervous to eat much breakfast and is already feeling quite hungry, thinks he said ‘There will be lunch at 11:30’. Later, in the staff room, Matti, the head teacher explains to some of her colleagues that one of the new girls was in tears in her office complaining that the Geography lesson was about strangely shaped countries which were difficult to understand and there was no lunch when she went to the dining hall. Matti says, “She thought Tom said ‘Geography at 11’ and Bill said ‘Lunch at 11:30’. And to add to the confusion, she thought they were the same person. Poor wee thing, she’s had a difficult day.”

In (62) it seems natural that Matti should use her names for the two teachers rather than Sam’s name “Mr Smith” for both of them when reporting Sam’s beliefs.

In summary, our approach to intensional constructions in natural language has two main components. Firstly, we use (hyper)intensional types rather than sets of possible worlds or situations as the objects of intensional predicates (like ‘believe’). Secondly, we characterize the truth-conditions of these constructions in terms of matching these types against other types (such as types characterizing the long-term memories of the believer, the reporter or the hearer of the report or, in the case of *illegal*, types characterizing a particular canon of law). This opens up possibilities for varying interpretations depending on both which types we match against and what kind of match is required. This makes the interpretation of intensional constructions much more context dependent than is normally assumed and interactive in the sense that we are often comparing (our view of) resources available to different agents.

In the rest of this section we will look at how these ideas can be applied to other intensional constructions that Montague (1973) treated: intensional transitive verbs, verbs taking inifinitival complements and intensional adverbs.

Our basic strategy for recasting Montague’s analysis of transitive verbs in terms of TTR is to treat them in terms of a predicate whose arguments are an individual (of type *Ind*) and a quantifier (of type *Quant*).

We have so far characterized the semantics of transitive verbs as in (63).

(63) If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}, \mathit{Ind} \rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \lambda\mathcal{Q}{:}\textit{Quant}\ .\ \ulcorner\lambda r_1{:}\big[\text{x}{:}\textit{Ind}\big]\ .\ \mathcal{Q}(\ulcorner\lambda r_2{:}\big[\text{x}{:}\textit{Ind}\big]\ .\ \big[\ \text{e}\ :\ p(r_1.\text{x}, r_2.\text{x})\ \big]\urcorner)\urcorner\urcorner$$

To this we can add a case for predicates with arity $\langle \textit{Ind}, \textit{Quant}\rangle$ given in (64).

(64) If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \textit{Ind}, \textit{Quant}\rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \lambda\mathcal{Q}{:}\textit{Quant}\ .\ \ulcorner\lambda r{:}\big[\text{x}{:}\textit{Ind}\big]\ .\ \big[\ \text{e}\ :\ p(r.\text{x}, \mathcal{Q})\ \big]\urcorner)\urcorner$$

One way to use this is to treat extensional verbs like *find* as based on predicates which have arity $\langle \textit{Ind}, \textit{Ind}\rangle$ and intensional verbs lik *seek* as based on predicates with arity $\langle \textit{Ind}, \textit{Quant}\rangle$.

An equivalent alternative is to follow Montague's treatment more closely and treat extensional verbs like *find* also as based on predicates with arity $\langle \textit{Ind}, \textit{Quant}\rangle$ but then to have an additional requirement, given in (65), for any predicate p which we want to be extensional, relating it to another predicate $p^\dagger$ which has arity $\langle \textit{Ind}, \textit{Ind}\rangle$

(65) We restrict attention to modal systems, $\mathbb{T}$, such that

$$p(a, \mathcal{Q}) \approx_{\mathbb{T}} \mathcal{Q}(\ulcorner\lambda r{:}\big[\text{x}{:}\textit{Ind}\big]\ .\ \big[\text{e}{:}p^\dagger(a, r.\text{x})\big]\urcorner)$$

This corresponds more or less exactly to Montague's meaning postulate in Montague (1973) for extensional transitive verbs (meaning postulate 4). In our version, however, the postulate is situation specific. If p is the predicate 'find', it says that any situation which is of the type 'find(a, Q)' is itself of the type obtained by "quantifying in" Q over a type constructed from the predicate 'find†' as specified in (65) and furthermore, as it is a biconditional any situation of the second type with the quantifier "exported" will also be of the first type with the quantifier as the second argument to 'find'.

In addition to this meaning postulate for extensional verbs, Montague also had a specific meaning postulate relating *seek* to *try to find* (his meaning postulate 9 in Montague, 1973). We will treat this rather differently in terms of what it means for a search to be successful. The intuitive idea is that a search is successful if you find what you are looking for. Our postulate is presented in (66), which uses the predicate 'successful' with arity $\langle \textit{Type}\rangle$.

(66) We restrict attention to modal systems, $\mathbb{T}$, such that

$$\text{successful}(\text{seek}(a, \mathcal{Q})) \sqsubseteq_{\mathbb{T}} \text{find}(a, \mathcal{Q})$$

This says that any situation is successful as a seeking of $\mathcal{Q}$ by a is also a situation in which a finds $\mathcal{Q}$.Note that this is a subtyping but not an equivalence. It could be that a finds Q without looking for it. Finding something does not always represent a successful search but you cannot have a successful search without finding what you are looking for. An event that witnesses your successful search for a unicorn would have to be one in which you find a unicorn.

Doing things this way gives us a rather different perspective on the relationship between extensional and intensional verbs than Montague's analysis. Montague treats both intensional and extensional verbs as relations between individuals and quantifier intensions. Extensional verbs are extensional in virtue of the fact that they are associated with a meaning postulate which says that there is an equivalence between the relation holding between some individual and the quantifier intension and the quantifier having wide scope over a formula which involves the corresponding relation between individuals. This can seem unintuitive in that the extensional case, which intuitively seems simpler than the intensional case, involves an extra meaning postulate which is not available for the intensional case. Thus the intensional case is taken as the basic case and the extensional case involves an additional inference. On the kind of analysis we are proposing both extensional and intensional verbs involve inferences whereby the quantifier is given wide scope. In the case of an extensional verb the inference is more direct and in some sense simpler. As exemplified by (65), it involves an equivalence and is an inference concerning the same situation which is of the type with the quantifier in argument position. The predicate involved is intuitively a version of the same predicate which takes two individuals as arguments rather than an individual and a quantifier. The inference involved with intensional verbs (as illustrated by (66)) is, however, more complex. Firstly, it does not involve an inference directly from the intensional verb holding between an individual and a quantifier, but rather concerning what would be a successful outcome of a situation of that type. Secondly, it involves a condiitional inference rather than an equivalence, though perhaps it is unclear whether that has anything to do with intuitive complexity. And thirdly, the conclusion does not involve the original intensional predicate but a distinct extensional predicate from which one can draw a conclusion with the quantifier exported. It seems then that in an intuitive sense something more complex, or at least special, is going on in the case of the intensional predicates.

In the PTQ fragment (Montague, 1973) *worship* was treated as an intensional verb. Early on in the literature on formal semantics this was generally regarded as a mistake (Bennett, 1974). It is indeed the case that a sentence like (67) does not entail the existence of a god.

(67) Kim worshipped a god

But on the other hand it seems that there has to be a specific (though possibly non-existent) god which Kim worshipped. This is different to the case with a true intensional verb like *seek*, *look for* or *need* where there is no requirement of specificity. The verb *worship* requires specificity but not existence of the object in this case. There are verbs which require existence but not specificity. An example of this is *book* as used in connection with booking a table at a restaurant. Compare the examples in (68), for example.

(68) a. # Kim booked a table but there were no tables
b. Kim was looking for / needed a table but there were no tables

(68a) seems inconsistent. If Kim booked a table then there must have been at least one table. (68b) on the other hand seems fine. Note, however, that you can book a table without booking a specific table. It may be that when you get to the restaurant there are several available tables and you get to choose which one you want to sit at. Booking a table can just mean that there will be a table available for you at the agreed time at the restaurant, not that any specific table has been reserved for you, although that, of course, is possible too and you may have specified which table you want (the one in the corner by the window). The verb *book* has thus the hallmarks of an intensional verb when it comes to specificity but nevertheless requires existence.

Let us deal first with *worship* and religious beliefs. Suppose that somebody says (69).

(69) Kim worships Zeus

This does not commit the speaker to the existence of Zeus, but is does commit the speaker to a system of religious belief in which there is a god Zeus and also commits her to the claim that Kim subscribes to such a belief system. Thus (70a) seems perfectly consistent whereas (70b) seems either inconsistent or at best to force a rather special meaning for *worship*.

(70) a. Kim worships Zeus, but I don't believe in Zeus
b. # Kim worships Zeus, but she doesn't believe in Zeus

If Kim worships Zeus, then she not only has to believe in Zeus, but she also has to believe that she worships Zeus. (71a) sounds strange or at least forces us into an unusual meaning for *worship*. In contrast (71b) with a standard extensional verb seems perfectly consistent.

(71) a. # Kim worships Zeus, but she doesn't believe that she worships Zeus
b. Kim found Harry, but she doesn't believe that she found Harry (because he was wearing a disguise)

This is another aspect of the content of *worship* which seems to suggest intensionality (or intentionality, with a 't'): it describes a conscious aspect of somebody's mental state. The verb *find*, on the other hand, can describe an external fact about an agent of which the agent itself is not aware.

We suppose that in our characterization of an agent's mental state by a type we can isolate a type that characterizes the agent's religious beliefs. We might regard this as a part of long term memory or as a separate component at the same level as long term memory in the type characterizing the agent's total information state, visual field and so on. The type corresponding to religious beliefs represents the way the world would be if the agent's religious beliefs were true. In (72) we give a type that could correspond to Kim's religious beliefs.

$$(72)\quad \begin{bmatrix} \mathrm{id}_1 & : & \begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & \mathrm{named(x, “Zeus”)} \end{bmatrix} \\ \mathrm{id}_2 & : & \begin{bmatrix} \mathrm{e} & : & \mathrm{god}(\Uparrow\mathrm{id}_1.\mathrm{x}) \end{bmatrix} \\ \mathrm{id}_3 & : & \begin{bmatrix} \mathrm{e} & : & \mathrm{worship}^\dagger(\mathrm{kim}, \Uparrow\mathrm{id}_1.\mathrm{x}) \end{bmatrix} \end{bmatrix}$$

Notice that the worship predicate we use is 'worship†' representing a relation between individuals. In this way *worship* is like the extensional verb *find* in that it is related by postulate to a †-variant of the same predicate, getting us the specificity. However, in the case of *worship* the type constructed with the †-predicate is embedded within another type representing religious belief which gives the verb an intensional quality.[10]

(69) commits the speaker not to the existence of Zeus but the type characterizing Kim's religious beliefs containing a match (in the sense we have discussed above) for the type (73).

$$(73)\quad \begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & \mathrm{named(x, “Zeus”)} \end{bmatrix}$$

In the sense that the object of *worship* has to be matched against a mental state just like the complement of *believe* rather than checked against the world *worship* is behaving like an intensional verb. It is also like an intensional verb in that we do not have a postulate like the one we have for *find* in (65). Where it is different from the classical cases of intensional verbs is that we do not have postulates like the one for *seek* in (66) but rather require a specific match for *Zeus* in the religious beliefs of the subject. (74) is a type corresponding to the speaker's long term memory which would fulfil the commitments of (69).

[10]Notice also that we are using 'kim' for the owner of the belief state, ignoring here *de se* issues.

(74) $$\left[\begin{array}{l} \text{id}_1{:}\left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Kim”)} \end{array}\right] \\ \text{id}_2{=}\left[\begin{array}{lll} \text{id}_1 & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Zeus”)} \end{array}\right] \\ \text{id}_2 & : & \left[\begin{array}{lll} \text{e} & : & \text{god}(\Uparrow\text{id}_1.\text{x}) \end{array}\right] \\ \text{id}_3 & : & \left[\begin{array}{lll} \text{e} & : & \text{worship}^\dagger(\Uparrow^2\text{id}_1.\text{x}, \Uparrow\text{id}_1.\text{x}) \end{array}\right] \end{array}\right]{:}\textit{RecType} \\ \text{id}_3{:}\left[\begin{array}{lll} \text{e} & : & \text{rbelieve}(\Uparrow\text{id}_1.\text{x}, \Uparrow\text{id}_2) \end{array}\right] \end{array}\right]$$

In the ‘id_3’-field here we are using the predicate ‘rbelieve’. Informally, rbelieve(a, T) is non-empty just in case a has T as a religious belief, that is the type identified as representing a’s religious beliefs in the type corresponding to her total information state is a subtype of T.

Armed with this view of the characterization of religious belief in information states we formulate a postulate for ‘worship’ in (75) which shows the intensionality and specificity requirements.

(75) e : worship(a,$\mathcal{Q}$) iff for some T

1. e : rbelieve(a,T)
2. $T \sqsubseteq_{\rightsquigarrow} \mathcal{Q}(\lambda r{:}\left[\text{x}{:}\textit{Ind}\right] .\ \text{worship}^\dagger(a,r.\text{x}))$

Clause (1) of (75) represents the intensionality (or perhaps more appropriately *intentionality*) requirement in that it requires us to use a type which characterizes the religious belief of the first argument of ‘worship’. Clause (2) represents the specificity requirement in that it requires the religious belief type to be a subtype of a type (possibly relabelled) obtained by “exporting” the quantifier Q.

Things are, however, a little more complicated than this. Jupiter is the Roman name for the god which in Greek is called Zeus.[11] Suppose that Kim is Greek oriented and does not know the name Jupiter. Suppose further that the speaker is Roman oriented and reports (76).

(76) Kim worships Jupiter

It seems that the speaker of (76) can be said to have spoken the truth even though Kim does not know the name Jupiter and the speaker does not believe that Jupiter exists or have any kind of religious belief in Jupiter. It is true because we know that Jupiter and Zeus are the same god in

[11] I am assuming for the sake of the example that *Jupiter* and *Zeus* are different names for the very same god and not names for distinct Roman and Greek gods who play similar roles in their respective pantheons. Ellen Breitholtz has pointed out to me that the presentation in the example may not correspond to standard views of Greek and Roman mythology.

the Roman-Greek pantheon. How can this be when the speaker is not committed to the existence of Zeus/Jupiter?

First consider that the speaker might be committed to a type characterizing part of the Roman pantheon, for example, (77).

(77)
$$\left[\begin{array}{lll} \mathrm{id}_1 & : & \left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & \text{named(x, “Jupiter”)} \end{array}\right] \\ \mathrm{id}_2 & : & \left[\begin{array}{lll} \mathrm{e} & : & \text{chief\_god}(\Uparrow\mathrm{id}_1.\mathrm{x}) \end{array}\right] \end{array}\right]$$

This can be incorporated into (74) as in (78).

(78)
$$\left[\begin{array}{ll} \mathrm{id}_1{:} & \left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & \text{named(x, “Kim”)} \end{array}\right] \\ \mathrm{id}_2{=} & \left[\begin{array}{lll} \mathrm{id}_1 & : & \left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & \text{named(x, “Zeus”)} \end{array}\right] \\ \mathrm{id}_2 & : & \left[\begin{array}{lll} \mathrm{e} & : & \text{god}(\Uparrow\mathrm{id}_1.\mathrm{x}) \end{array}\right] \\ \mathrm{id}_3 & : & \left[\begin{array}{lll} \mathrm{e} & : & \text{worship}^{\dagger}(\Uparrow^2\mathrm{id}_1.\mathrm{x}, \Uparrow\mathrm{id}_1.\mathrm{x}) \end{array}\right] \end{array}\right] {:}\mathit{RecType} \\ \mathrm{id}_3{:} & \left[\begin{array}{lll} \mathrm{e} & : & \text{rbelieve}(\Uparrow\mathrm{id}_1.\mathrm{x}, \Uparrow\mathrm{id}_2) \end{array}\right] \\ \mathrm{id}_4{=} & \left[\begin{array}{lll} \mathrm{id}_1 & : & \left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & \text{named(x, “Jupiter”)} \end{array}\right] \\ \mathrm{id}_2 & : & \left[\begin{array}{lll} \mathrm{e} & : & \text{chief\_god}(\Uparrow\mathrm{id}_1.\mathrm{x}) \end{array}\right] \end{array}\right] {:}\mathit{RecType} \\ \mathrm{id}_5{:} & \left[\begin{array}{lll} \mathrm{e} & : & \text{roman\_pantheon}(\Uparrow\mathrm{id}_4) \end{array}\right] \end{array}\right]$$

In (78) no connection is expressed between the types in the ‘id_2’-field and the ‘id_4’-field. Note that they are, however, aligned in their labelling. The individual named Zeus and the individual named Jupiter are required to have the same labelling in witnesses for the two types and similarly for the situations that show they have those names respectively as well as the situations that show the individual is a god and chief god respectively. We shall take this alignment of labelling of the two types to be a significant aspect of the type of information state that (78) represents. We shall say that the type in the ‘id_4’-field is a point of view on the type in the ‘id_2’-field. It represents the agent’s alternative perspective on certain aspects of what she believes to be Kim’s religious beliefs. We introduce an additional field to (78) in order to represent this connection between the two types and we reorganize the point of view and Kim’s religious beliefs into a single record type since it will be important to be able to refer to a single situation providing this information in computing the semantics of *worship*. This is given in (79).

(79)
$$\left[\begin{array}{ll}
\text{id}_1{:} & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Kim”)} \end{array}\right] \\
\text{id}_2{:} & \left[\begin{array}{lll}
\text{id}_1{=} & \left[\begin{array}{lll} \text{id}_1 & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Zeus”)} \end{array}\right] \\ \text{id}_2 & : & \left[\begin{array}{lll} \text{e} & : & \text{god}(\Uparrow\text{id}_1.\text{x}) \end{array}\right] \\ \text{id}_3 & : & \left[\begin{array}{lll} \text{e} & : & \text{worship}^{\dagger}(\Uparrow^{3}\text{id}_1.\text{x}, \Uparrow\text{id}_1.\text{x}) \end{array}\right] \end{array}\right] & : \ \textit{RecType} \\
\text{id}_2{:} & \left[\begin{array}{lll} \text{e} & : & \text{rbelieve}(\Uparrow^{2}\text{id}_1.\text{x}, \Uparrow\text{id}_1) \end{array}\right] & \\
\text{id}_3{=} & \left[\begin{array}{lll} \text{id}_1 & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Jupiter”)} \end{array}\right] \\ \text{id}_2 & : & \left[\begin{array}{lll} \text{e} & : & \text{chief_god}(\Uparrow\text{id}_1.\text{x}) \end{array}\right] \end{array}\right] & : \ \textit{RecType} \\
\text{id}_4{:} & \left[\begin{array}{lll} \text{e} & : & \text{roman_pantheon}(\Uparrow\text{id}_3) \end{array}\right] & \\
\text{id}_5{:} & \left[\begin{array}{lll} \text{e} & : & \text{pov}(\Uparrow\text{id}_3, \Uparrow\text{id}_1) \end{array}\right] &
\end{array}\right]
\end{array}\right]$$

The idea of a point of view is that it represents an alternative take on certain aspects of another type. As before we can obtain a complete alternative version of the original type by taking the asymmetric merge of the original type with the point of view. Thus in the case of the relevant types in (79) we can obtain (80a) which is identical with (80b).

(80) a.
$$\left[\begin{array}{lll} \text{id}_1 & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Zeus”)} \end{array}\right] \\ \text{id}_2 & : & \left[\begin{array}{lll} \text{e} & : & \text{god}(\Uparrow\text{id}_1.\text{x}) \end{array}\right] \\ \text{id}_3 & : & \left[\begin{array}{lll} \text{e} & : & \text{worship}^{\dagger}(\text{kim}, \Uparrow\text{id}_1.\text{x}) \end{array}\right] \end{array}\right] \boxed{\dot{\wedge}} \left[\begin{array}{lll} \text{id}_1 & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Jupiter”)} \end{array}\right] \\ \text{id}_2 & : & \left[\begin{array}{lll} \text{e} & : & \text{chief\_god}(\Uparrow\text{id}_1.\text{x}) \end{array}\right] \end{array}\right]$$

b.
$$\left[\begin{array}{lll} \text{id}_1 & : & \left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{named(x, “Jupiter”)} \end{array}\right] \\ \text{id}_2 & : & \left[\begin{array}{lll} \text{e} & : & \text{chief\_god}(\Uparrow\text{id}_1.\text{x}) \end{array}\right] \\ \text{id}_3 & : & \left[\begin{array}{lll} \text{e} & : & \text{worship}^{\dagger}(\text{kim}, \Uparrow\text{id}_1.\text{x}) \end{array}\right] \end{array}\right]$$

We will call (80) a *complete point of view*. We will now allow two alternative witness conditions for ‘worship’. The first, as before, checks that the type corresponding to the religious beliefs of the subject is a subtype of the type resulting from exporting the quantifier. The second checks that a complete point of view fulfils this condition. This is parallel to the witness conditions we gave for ‘believe’. The two witness conditions are given in (81).

(81) a. e : worship(a,Q) if for some T

1. e : rbelieve(a,T)
2. $T \sqsubseteq_{\leadsto} Q(\lambda r{:}\left[\text{x}{:}\textit{Ind}\right]\ .\ \text{worship}^{\dagger}(a,r.\text{x}))$

b. e : worship(a,Q) if for some T_1, T_2

1. $e :_{\underline{\varepsilon}}$ rbelieve(a, T_1)
2. $e :_{\underline{\varepsilon}}$ pov(T_2, T_1)
3. $T_1 \boxed{\wedge} T_2 \sqsubseteq_{\rightsquigarrow} Q(\lambda r{:}\left[\text{x:}\mathit{Ind}\right] \text{ . worship}^{\dagger}(a,r.\text{x}))$

We will allow the background conditions of proper names, for example, the type in (82) to match a complete resource and indeed (82) matches (80).

(82) $\left[\begin{array}{lll} \text{x} & : & \mathit{Ind} \\ \text{e} & : & \text{named(x, “Jupiter”)} \end{array}\right]$

Now consider the case with the indefinite article. Suppose we have a situation of the type (74), where Kim worships Zeus, who is, according to Kim's religious beliefs, a god. In order to show that Kim workships a god we may show, following (81a), that this type is a subtype of (83)[12].

(83) exist(god′, $\lambda r{:}\left[\text{x:}\mathit{Ind}\right]$. worship†(kim,r.x))

In virtue of the witness conditions associated with ‘exist’ introduced in Chapter 3, example (61), the subtype relation holds between (74) and (83). That is, any situation of type (74) will be of type (83) since in such a situation there will be a individual who is a god whom Kim worships.

Now let us consider a case which shows that we can use the witness condition (81b). In (84) it is presumably not the case that it is part of Kim's religious beliefs that the god she worships is a false god.

(84) Kim worships a false god

Rather the phrase *false god* can be used to represent a point of view on the part of the speaker. In (85) we add such a point of view to (79).

[12]Where ‘god′’ abbreviates $\lambda r{:}\left[\text{x:}\mathit{Ind}\right]$. $\left[\text{e:god}(r.\text{x})\right]$.

$$
(85)\quad \left[\begin{array}{ll}
\text{id}_1: & \left[\begin{array}{lll} \text{x} & : & \textit{Ind}\\ \text{e} & : & \text{named(x, “Kim”)}\end{array}\right]\\
\text{id}_2: & \left[\begin{array}{ll}
\text{id}_1= & \left[\begin{array}{lll} \text{id}_1 & : & \left[\begin{array}{lll}\text{x}&:&\textit{Ind}\\ \text{e}&:&\text{named(x, “Zeus”)}\end{array}\right]\\ \text{id}_2 & : & \left[\begin{array}{lll}\text{e}&:&\text{god}(\Uparrow\text{id}_1.\text{x})\end{array}\right]\\ \text{id}_3 & : & \left[\begin{array}{lll}\text{e}&:&\text{worship}^\dagger(\Uparrow^3\text{id}_1.\text{x}, \Uparrow\text{id}_1.\text{x})\end{array}\right]\end{array}\right] \quad : \quad \textit{RecType}\\
\text{id}_2: & \left[\begin{array}{lll}\text{e}&:&\text{rbelieve}(\Uparrow^2\text{id}_1.\text{x}, \Uparrow\text{id}_1)\end{array}\right]\\
\text{id}_3= & \left[\begin{array}{lll} \text{id}_1 & : & \left[\begin{array}{lll}\text{x}&:&\textit{Ind}\\ \text{e}&:&\text{named(x, “Jupiter”)}\end{array}\right]\\ \text{id}_2 & : & \left[\begin{array}{lll}\text{e}&:&\text{chief\_god}(\Uparrow\text{id}_1.\text{x})\end{array}\right]\end{array}\right] \quad : \quad \textit{RecType}\\
\text{id}_4: & \left[\begin{array}{lll}\text{e}&:&\text{roman\_pantheon}(\Uparrow\text{id}_3)\end{array}\right]\\
\text{id}_5: & \left[\begin{array}{lll}\text{e}&:&\text{pov}(\Uparrow\text{id}_3, \Uparrow\text{id}_1)\end{array}\right]\\
\text{id}_6= & \left[\begin{array}{lll} \text{id}_1 & : & \left[\begin{array}{lll}\text{x}&:&\textit{Ind}\\ \text{e}&:&\text{named(x, “Zeus”)}\end{array}\right]\\ \text{id}_2 & : & \left[\begin{array}{lll}\text{e}&:&\text{false\_god}(\Uparrow\text{id}_1.\text{x})\end{array}\right]\end{array}\right] \quad : \quad \textit{RecType}\\
\text{id}_7: & \left[\begin{array}{lll}\text{e}&:&\text{pov}(\Uparrow\text{id}_6, \Uparrow\text{id}_1)\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

One thing to note about (85) is that an agent can have more than one distinct point of view on the same type, here shown by the ‘$\text{id}_2.\text{id}_3$’-field and ‘$\text{id}_2.\text{id}_6$’-field which both introduce points of view on the type in the ‘id_2’-field. In searching for a match for the content of (84) we may use the asymmetric merge of the ‘$\text{id}_2.\text{id}_1$’ type with either the ‘$\text{id}_2.\text{id}_3$’ type or the ‘$\text{id}_2.\text{id}_6$’ type. That is, there will be a relabelling of the content of (84), such that any situation of type (85) will also be a situation of the type which is the relabelled content. To make this concrete consider a putative (non-parametric) content for (84) represented in (86a) and the relabelling given in (86b), yielding the relabelled type in (86c).

$$
(86)\quad \text{a.}\quad \left[\begin{array}{lll}
\text{x} & : & \textit{Ind}\\
\text{c} & : & \text{named(x,“Kim”)}\\
\text{e} & : & \text{worship(x, } \lambda P{:}\textit{Ppty}\,.\,\left[\text{e:exist(false\_god}', P)\right])
\end{array}\right]
$$

$$
\begin{array}{ll}
\text{b.} & \text{x} \rightsquigarrow \text{id}_1.\text{x}\\
& \text{c} \rightsquigarrow \text{id}_1.\text{e}\\
& \text{e} \rightsquigarrow \text{id}_2
\end{array}
$$

$$
\text{c.}\quad \left[\begin{array}{lll}
\text{id}_1 & : & \left[\begin{array}{lll}\text{x} & : & \textit{Ind}\\ \text{e} & : & \text{named(x,“Kim”)}\end{array}\right]\\
\text{id}_2 & : & \text{worship(x, } \lambda P{:}\textit{Ppty}\,.\,\left[\text{e:exist(false\_god}', P)\right])
\end{array}\right]
$$

We can see that any situation of type (85) would also be of type (86c) given the witness conditions associated with ‘worship’ and ‘exist’ that we have discussed.

Let us now consider sentences such as *Kim booked a table* where there is no inference to a specific table but nevertheless an inference to the existence of a table. This can be achieved by introducing the postulate in (87).

(87) We restrict our attention to model type systems, $\mathbb{T}$, such that if a:*Ind* and Q is a monotone increasing quantifier, then

$$\text{book}(a,Q) \sqsubseteq_{\mathbb{T}} Q(\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix} \;.\; \begin{bmatrix}\text{e:be}(r.\text{x})\end{bmatrix})$$

Intuitively, (87) requires that if there is some situation, s, in which a books a table, then there is some table in s which is a component of some situation (following the witness conditions for ‘be’ presented in Chapter 3, p. 133). Notice that we do not get such an inference if the quantifier is monotone decreasing. Thus *Kim booked no table* (to the extent that this is an acceptable sentence of English) does not imply that there are no tables and neither does it imply that there are tables. We will discuss monotonicity in Chapter 7.

We now turn our attention to a phenomenon first discussed in the semantics literature by Fodor (1970), p. 226ff. She points out that (88) can mean that Charley does not want a specific coat even if Charley would not describe what he wants as a “coat like Bill’s”.

(88) Charley wants to buy a coat like Bill’s

The sentence could be true on what Montague would call a *de dicto* reading even though Charley does not know Bill, or a least does not know what kind of coat he has. This kind of example, seems straightforwardly treatable with the notion of point of view that we have developed. Consider the type (89), representing the long term memory of some agent.

(89)
$$\begin{bmatrix}
\text{id}_1{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:named(x,“Charley”)}\end{bmatrix} \\
\text{id}_2{:}\begin{bmatrix}
\text{id}_1{=}\begin{bmatrix}\text{id}_1{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:coat(x)}\end{bmatrix}\\ \text{id}_2{:}\begin{bmatrix}\text{e:trenchcoat}(\Uparrow\text{id}_1.\text{x})\end{bmatrix}\\ \text{id}_3{:}\begin{bmatrix}\text{e:has_big_pockets}(\Uparrow\text{id}_1.\text{x})\end{bmatrix}\\ \text{id}_4{:}\begin{bmatrix}\text{e:buy}^{\dagger}(\Uparrow^3\text{id}_1.\text{x},\ \Uparrow\text{id}_1.\text{x})\end{bmatrix}\end{bmatrix}{:}\mathit{RecType} \\
\text{id}_2{:}\text{want}^{\dagger}(\Uparrow^2\text{id}_1.\text{x},\ \text{id}_1) \\
\text{id}_3{=}\begin{bmatrix}\text{id}_1{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:coat(x)}\end{bmatrix}\\ \text{id}_5{:}\begin{bmatrix}\text{e:coat_like_Bill's}(\Uparrow\text{id}_1.\text{x})\end{bmatrix}\end{bmatrix}{:}\mathit{RecType} \\
\text{id}_4{:}\text{pov}(\text{id}_3,\ \text{id}_1)
\end{bmatrix}
\end{bmatrix}$$

Here we use the predicate ‘buy†’ which is the buy-relation between individuals and the predicate ‘want†’ which has arity $\langle$*Ind*, *RecType*$\rangle$. It is related to two other want-predicates: ‘want$_P$’ with arity $\langle$*Ind*, *Ppty*$\rangle$ and ‘want$_Q$’ with arity $\langle$*Ind*, *Quant*$\rangle$ used to treat examples like, respectively, *want to buy a coat* and *want a coat*. These two predicates are related to ‘want†’ as shown in (90).

(90) We restrict our attention to modal systems, $\mathbb{T}$, such that

a. if a : *Ind* and P : *Ppty*, then

$$\text{want}_P(a, P) \approx_{\mathbb{T}} \text{want}^\dagger(a, P(\left[\text{x}{=}a\right]))$$

b. if a : *Ind* and Q : *Quant*, then

$$\text{want}_Q(a,Q) \approx_{\mathbb{T}} \text{want}^\dagger(a, Q(\ulcorner\lambda r{:}\left[\text{x}{:}\textit{Ind}\right] \,.\, \text{have}(a, r.\text{x})\urcorner))$$

In order to characterize witness conditions for ‘want†’ we need to assume that an agent’s total information state include a record type representing the agent’s desires, parallel to long term memory and religious beliefs. Intuively the type for desires is a type which represents the way the world would be if the agent’s desires were fulfilled. Thus a total information state would belong to a subtype of (91).

$$(91)\quad \begin{bmatrix} \text{ltm} & : & \textit{RecType} \\ \text{rbel} & : & \textit{RecType} \\ \text{des} & : & \textit{RecType} \end{bmatrix}$$

We will introduce a predicate ‘des’ which holds between an individual and a record type (that is, with arity $\langle$*Ind*,*RecType*$\rangle$). e : des(a, T) just in case T is a’s desires in e. This should require that if r is a’s information state then r.des $= T$.

Now we can characterize witness conditions for ‘want†’ as given in (92).

(92) e : want$^\dagger(a, T)$ if for some T'

e : des(a, T')

and $T' \sqsubseteq_{\leadsto} T$

e : want$^\dagger(a, T)$ if for some T_1 and T_2

$e :_{\underline{\varepsilon}}$ want$^\dagger(a, T_1)$

$e :_{\underline{\varepsilon}}$ pov(T_2, T_1)

and $T_1 \boxed{\dot{\wedge}} T_2 \sqsubseteq_{\leadsto} T$

Our analysis of Fodor’s example is similar to Schwager (2009) in that the analysis using a point of view and asymmetric merge involves replacing part of the original content of the attitude with something from the perspective of the speaker. Schwager[13] introduces what she calls the *replacement principle* quoted in (93).

(93) For the sake of reporting an attitude, a property that is involved in the content of the attitude that is to be reported (the **reported property**) can be replaced by a different property (the **reporting property**) as long as the reported property is a subset of the reporting property at all relevant worlds.

(Schwager, 2009, p. 409)

This addresses the important question of what replacements we can make. As Schwager points out we cannot make arbitrary replacements and use them to report an attitude. For example, we cannot report Charley's desire to buy a trenchcoat with big pockets by (94).

(94) Charley wants to buy a unicorn

For us, this question concerns what must hold if the 'pov'-relation holds between two types. We could imitate something like Schwager's replace principle by requiring (95).

(95) If pov(T_1, T_2) is witnessed then $T_2 \sqsubseteq T_2 \boxed{\wedge} T_1$

This says that if the world is of the original attitude type then it is also of the type resulting from asymmetrically merging the attitude type with the point of view. This successfully rules out replacing a trenchcoat with big pockets with a unicorn. If the world is such that Charley has a trenchcoat with big pockets it does not follow that he has a unicorn.

Unfortunately, this constraint does not seem to hold for all of the examples for which we have suggested using points of view. For example, if the original attitude type involves two heavenly bodies, Hesperus and Phosphorus, which rise respectively in the evening and the morning and the point of view requires Hesperus and Phosphorus to correspond to one heavenly body, Venus which rises in the morning and the evening, then it is not the case that any situation which contains the two heavenly bodies would be one in which there is only one heavenly body (and not *vice versa* either). There is no subtyping relation here, just a disagreement about how many heavenly bodies are involved and what they are called.

Similarly, consider the example where, according to the original attitude, Kim worships a god called Zeus and according to the point of view Kim worships a false god called Zeus. It is not obvious that a situation in which the first holds is also a situation in which the second holds or

[13] currently named Magdalena Kaufmann

vice versa, although one might argue that this depends on whether false gods are gods or not. I would tend to think that sometimes we make the inference from *false god* to *god* and sometimes we do not and that this is part of the general nature of semantic flux in language. However, an inference from *god* to *false god* seems unlikely. Again the point of view seems to reflect a disagreement about the status of Zeus rather than a subtyping relation.

A potential conflict in judgement between the reporter and the attitude bearer also arises in Fodor examples like those in (96).

(96) a. Charley wants to buy something nonexistent

b. Charley wants to buy an uncool coat

If Charley's attitude involves a trenchcoat with big pockets, it may in fact be the case that there are no such trench coats but that does not mean that the type *Trenchcoat with big pockets* is a subtype of *Nonexistent*. Recall that for T_1 to be a subtype of T_2 it has to be the case that no matter what we assign to basic types and ptypes (that is, no matter which possibility we consider), something of type T_1 would also be of type T_2. But there are presumably possibilities in which there are trenchcoats with big pockets. What (96a) seems to commit the speaker to is that there are no trenchcoats with big pockets in the actual possibility we are considering, not that trenchcoats with big pockets are impossible. In actual fact the speaker seems to be committed to a falsehood here because trenchcoats in general have big pockets. We cannot say the same about (96b). However, cool we may think trenchcoats are the speaker is entitled to her opinion. The word *uncool* is a predicate of personal taste and the concept of *faultless disagreement* (Kölbel, 2004) becomes relevant. For a suggestion of how predicates of personal taste could be treated using TTR and some references to other literature see Cooper (2015). What this points to is that the speaker is replacing a judgement by the attitude bearer with a judgement of her own. This could be expressed by introducing the action rule in (97).

(97) $$\frac{:_A \text{pov}(T_1, T_2) \qquad s :_B T_2}{s :_A T_2 \boxed{\underset{\cdot}{\wedge}} T_1}$$

(97) says that if A judges pov(T_1, T_2) to be witnessed (that is, A judges that T_1 is a point of view of T_2) and B judges s to be of type T_2, then A is licensed (afforded) to judge s to be of the type resulting from asymmetrically merging T_2 with T_1. Note that this action rule concerns the judgements that two agents make rather than the way things actually are in the world. It belongs to the realm of our theory of action based on type theory rather than to the type theory itself. This means that in principle there is nothing preventing a speaker reporting Charley's desire to buy a trenchcoat with large pockets as a desire to buy a unicorn, provided that the speaker is willing to commit to a claim that she would judge a trenchcoat with large pockets to be a

unicorn, something that we would find unexpected given the normal meanings associated with these words.

The proposal here also has aspects in common with the proposal by Pross (ms) which presents a semantics in terms of DRT which takes account of how to represent the attitudes of an agent and analyzes the attitude report in terms of this. Our approach to representing the attitudes in TTR has a good deal in common with the DRT approach as set forth in Kamp (1990); Kamp *et al.* (2011) and developed in Maier (2016, 2017).

Where our proposal differs from previous proposals in the literature is that we are not concerned with trying to identify objects in possible worlds in order to get the Fodorean reading. Rather we are concerned with how different agents might judge situations of certain types. Whether there are situations of the relevant types is not a question which is of relevance to the analysis. Let us see how this relates to examples which have been discussed in the literature. In order to do this we will first say in a little more detail what the content of an utterance of *Charlie wants to buy a coat like Bill's* will be. Let us first look at the content of *Charlie bought a coat like Bill's* (ignoring issues of tense), given in (98).

(98) $$\lambda c{:}\left[\begin{array}{l} \textit{Cntxt} \\ \mathfrak{c}{:}\left[\begin{array}{l} \text{f}{:}\left[\begin{array}{l} \textit{PropCntxt} \\ \text{x}{:}\textit{Ind} \\ \text{e}{:}\text{named(x, “Charlie”)} \end{array}\right] \\ \text{a}{:}\textit{PropCntxt} \end{array}\right] \end{array}\right] . \left[\begin{array}{lll} \text{e} & : & \text{buy}(\mathfrak{c}.\text{f.x}, \textbf{a}\frown\textbf{coat\_like\_Bill's}(\text{c.a})) \end{array}\right]$$

Here we use ‘$\textbf{a}\frown\textbf{coat\_like\_Bill's}$’ to represent the content of an utterance of the noun phrase *a coat like Bill's*. (For simplicity of presentation we are ignoring the fact that *Bill* makes a similar contribution to the context type as *Charlie* though on the path headed by ‘a’. We are also considerably simplifying the structure of the context for presentational purposes.) Since ‘buy’ is an extensional predicate it will obey the constraint in (99) which relates ‘buy’ to the corresponding predicate with two individual arguments, ‘$\text{buy}^{\dagger}$’.

(99) We restrict our attention to modal type systems, $\mathbb{T}$, such that

$$\text{buy}(a{,}Q) \approx_{\mathbb{T}} Q(\lambda r{:}\left[\text{x}{:}\textit{Ind}\right] . \left[\begin{array}{lll} \text{e} & : & \text{buy}^{\dagger}(a, r.\text{x}) \end{array}\right])$$

This means that (98) is equivalent to (100), in the sense that any record of the former type will be of the latter type and *vice versa*.

(100) $\lambda c{:}\begin{bmatrix} \textit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix} \text{f:}\begin{bmatrix} \textit{PropCntxt} \\ \text{x:}\textit{Ind} \\ \text{e:named(x, "Charlie")} \end{bmatrix} \\ \text{a:}\textit{PropCntxt} \end{bmatrix} \end{bmatrix}.$
$\begin{bmatrix} \text{e} & : & \textbf{a}^\frown\textbf{coat\_like\_Bill's}(\text{c.a})(\lambda r{:}\begin{bmatrix}\text{x:}\textit{Ind}\end{bmatrix}.\begin{bmatrix} \text{e} & : & \text{buy}(\mathfrak{c}.\text{f.x}, r.\text{x})\end{bmatrix}) \end{bmatrix}$

(100) is (101) where we have spelled out the contribution of the indefinite article.

(101) $\lambda c{:}\begin{bmatrix} \textit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix} \text{f:}\begin{bmatrix} \textit{PropCntxt} \\ \text{x:}\textit{Ind} \\ \text{e:named(x, "Charlie")} \end{bmatrix} \\ \text{a:}\textit{PropCntxt} \end{bmatrix} \end{bmatrix}.$
$\begin{bmatrix} \text{e} & : & \begin{bmatrix} \text{range=}\textbf{coat\_like\_Bill's}(c.\text{a}) & : & \textit{Ppty} \\ \text{scope=}\lambda r{:}\begin{bmatrix}\text{x:}\textit{Ind}\end{bmatrix}.\begin{bmatrix}\text{e:buy}(\mathfrak{c}.\text{f.x}, r.\text{x})\end{bmatrix} & : & \textit{Ppty} \\ \text{e} & : & \text{exist(range, scope)} \end{bmatrix} \end{bmatrix}$

Now let us consider the non-parametric content of *Charlie wants to buy a coat like Bill's* given in (102a) which, by (90a), is equivalent to (102b).

(102) a. $\begin{bmatrix}\text{e:want}_P(\text{charlie}, \lambda r{:}\begin{bmatrix}\text{x:}\textit{Ind}\end{bmatrix}.\begin{bmatrix}\text{e:buy}(r.\text{x}, \lambda P{:}\textit{Ppty}.\begin{bmatrix}\text{e:exist(coat\_like\_Bill's}', P)\end{bmatrix})\end{bmatrix})\end{bmatrix}$

b. $\begin{bmatrix}\text{e:want}^\dagger(\text{charlie}, \begin{bmatrix}\text{e:buy(charlie}, \lambda P{:}\textit{Ppty}.\begin{bmatrix}\text{e:exist(coat\_like\_Bill's}', P)\end{bmatrix})\end{bmatrix})\end{bmatrix}$

Note that 'want†', in virtue of (92), introduces the possibility of a point of view.

We will now examine whether the kind of content expressed in (102a) can be matched to a number of scenarios for Fodorean readings which have been discussed in the literature. Here we follow the recent survey of the literature presented by Pross (ms) in Section 1.2 of his paper. Consider the scenario in (103).

(103) Suppose a store sells some jackets that all look like Malte's and that Adrian does not know anything about Malte. Assume further that Adrian wants one of those jackets and any of them is an option.

(Romoli and Sudo (2009))

In (104) we exhibit a type which corresponds to how the speaker might represent this scenario in memory.

(104)
$$\left[\begin{array}{ll} \mathrm{id}_1 : & \left[\begin{array}{l} \mathrm{x}:\mathit{Ind} \\ \mathrm{e}:\mathrm{named}(\mathrm{x}, \text{“Adrian”}) \end{array}\right] \\ \mathrm{id}_2 : & \left[\begin{array}{l} \mathrm{id}_1 = \left[\begin{array}{l} \mathrm{id}_1 = \mathrm{jacket\_in\_the\_shop}' : \mathit{Ppty} \\ \mathrm{id}_2 : \left[\mathrm{e}:\mathrm{buy}(\Uparrow^3\mathrm{id}_1.\mathrm{x}, \lambda P:\mathit{Ppty}\ .\ \left[\mathrm{e}:\mathrm{exist}(\Uparrow\mathrm{id}_1, P)\right])\right] \end{array}\right] : \mathit{RecType} \\ \mathrm{id}_2 : \mathrm{want}^\dagger(\Uparrow^2\mathrm{id}_1.\mathrm{x}, \mathrm{id}_1) \\ \mathrm{id}_3 = \left[\mathrm{id}_1 = \mathrm{jacket\_like\_Malte's}' : \mathit{Ppty}\right] : \mathit{RecType} \\ \mathrm{id}_4 : \mathrm{pov}(\mathrm{id}_3, \mathrm{id}_1) \end{array}\right] \end{array}\right]$$

(104) requires that Adrian's desire is to buy something with the property of being a jacket in the shop. The alternative point of view is that the property of being a jacket in the shop can be replaced with the property of being a jacket like Malte's. If the world matches this type then the sentence *Adrian wants to buy a jacket like Malte's* is true.

The next scenario Pross considers is (105).

(105) Suppose a store offers some jackets that all look like Malte's and that Adrian does not know anything about Malte. Assume that some of the jackets are on sale while others are not and that Adrian is aware of this. Assume further that Adrian wants one of the jackets on sale and any of them is an option.

The point of this is that it emphasizes that the property of jackets involved in Adrian's desire need not be coextensive with the property in the point of view. That is, it is still the case that all the jackets in the shop are like Malte's but Adrian has his sights set on a subset of them (those on sale) although he has not chosen any particular jacket among those. The type we exhibit for this scenario (in (106)) is almost exactly the same as the previous one.

(106)
$$\left[\begin{array}{ll} \mathrm{id}_1 : & \left[\begin{array}{l} \mathrm{x}:\mathit{Ind} \\ \mathrm{e}:\mathrm{named}(\mathrm{x}, \text{“Adrian”}) \end{array}\right] \\ \mathrm{id}_2 : & \left[\begin{array}{l} \mathrm{id}_1 = \left[\begin{array}{l} \mathrm{id}_1 = \mathrm{jacket\_on\_sale\_in\_the\_shop}' : \mathit{Ppty} \\ \mathrm{id}_2 : \left[\mathrm{e}:\mathrm{buy}(\Uparrow^3\mathrm{id}_1.\mathrm{x}, \lambda P:\mathit{Ppty}\ .\ \left[\mathrm{e}:\mathrm{exist}(\Uparrow\mathrm{id}_1, P)\right])\right] \end{array}\right] : \mathit{RecType} \\ \mathrm{id}_2 : \mathrm{want}^\dagger(\Uparrow^2\mathrm{id}_1.\mathrm{x}, \mathrm{id}_1) \\ \mathrm{id}_3 = \left[\mathrm{id}_1 = \mathrm{jacket\_like\_Malte's}' : \mathit{Ppty}\right] : \mathit{RecType} \\ \mathrm{id}_4 : \mathrm{pov}(\mathrm{id}_3, \mathrm{id}_1) \end{array}\right] \end{array}\right]$$

The only difference between this and the previous type is that we have replaced the property 'jacket_in_the_shop′' with 'jacket_on_sale_in_the_shop′'. Note that the constraint on 'pov' that we introduced in (97) did not require coextension in any way, only that if an agent were to judge a situation as of type T_1 then the agent with the point of view would judge the situation to be of the point of view type. This allows for the possibility that there could be additional situations of the point of view type which the first agent would not judge to be of type T_1. This view of things was in fact already important for the first scenario, since, while Adrian is focussed on the jackets in the shop, the speaker presumably could consider that there are other jackets in addition to those in the shop which are like Malte's. That is, the sentence would not be falsified by discovering a jacket like Malte's which is not in the shop.

Pross's third scenario is (107) which he offers as problematic for the proposal in von Fintel and Heim (2011) because there are no actual jackets like Malte's as would be required by their analysis.

(107) Suppose Adrian has seen a picture of a certain green Burberry jacket in a catalogue and wants to buy one. Unbeknownst to Adrian, Malte happens to own exactly such a green Burberry jacket. Unbeknownst to Adrian, the type of jacket in the picture which Adrian has seen is sold out and no further jackets of this type have been produced yet: there are no actual jackets like Malte's.

This scenario could correspond to the type in (108).

$$
(108)\quad \left[\begin{array}{ll}
\text{id}_1 : & \left[\begin{array}{l} \text{x}:\textit{Ind} \\ \text{e}:\text{named(x, “Adrian”)} \end{array}\right] \\
\text{id}_2 : & \left[\begin{array}{l}
\text{id}_1 = \left[\begin{array}{l} \text{id}_1=\text{jacket\_like\_the\_one\_in\_the\_catalogue}':\textit{Ppty} \\ \text{id}_2:\left[\text{e:buy}(\Uparrow^3\text{id}_1.\text{x},\ \lambda P:\textit{Ppty}\ .\ \left[\text{e:exist}(\Uparrow\text{id}_1,\ P)\right])\right] \end{array}\right]:\textit{RecType} \\
\text{id}_2:\text{want}^{\dagger}(\Uparrow^2\text{id}_1.\text{x},\ \text{id}_1) \\
\text{id}_3=\left[\text{id}_1=\text{jacket\_like\_Malte's}':\textit{Ppty}\right]:\textit{RecType} \\
\text{id}_4:\text{pov}(\text{id}_3,\ \text{id}_1)
\end{array}\right]
\end{array}\right]
$$

The only difference between this and the previous type is that we have replaced the property 'jacket_on_sale_in_the_shop′' with the property 'jacket_like_the_one_in_the_catalogue′'. Note that the constraint on 'pov' that we introduced in (97) does not require that anything have either the property 'jacket_like_the_one_in_the_catalogue′' or 'jacket_like_Malte's′', only that if an agent were to judge a situation as of the type involving the first property then the agent with the point of view would judge the situation to be of the type involving the second property. Whether there actually are such jackets is an independent question.

Pross (ms) introduces a further scenario for the Fodorian reading in Section 3.5 of his paper which we reproduce in (109).

(109) Adrian has seen a jacket which has three stripes on its sleeves and wants to buy such a jacket. However, he has read that Adidas uses child labour in the production of its jackets, so the additional condition for his purchase is that the jacket is not from Adidas. If Adrian does not know that Adidas is the brand with the three stripes, he has a desire that he

would paraphrase as “I want to buy a jacket from the brand with the three stripes but not from Adidas.” Fritz hears Adrian’s utterance and as he has seen Malte’s jacket which has three stripes and as he also knows about the problem with child labour and Adidas he believes that Malte would never buy a jacket which is made using child labour. Fritz also doesn’t know that Adidas is the brand with the three stripes. He reports Adrian’s desire as “Adrian wants to buy a jacket like Malte’s”.

This mixes in the problem of contradictory beliefs with that of Fodorean readings. The type corresponding to Fritz’s information state could be represented by the type in (110).

(110)
$$\left[\begin{array}{ll}
\mathrm{id}_1: & \left[\begin{array}{l}\mathrm{x}{:}\mathit{Ind}\\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Adrian”})\end{array}\right]\\
\mathrm{id}_2: & \left[\begin{array}{l}
\mathrm{id}_1{=}\left[\begin{array}{l}\mathrm{id}_1{=}\mathrm{jacket_with_three_stripes_on_its_sleeves}'{:}\mathit{Ppty}\\ \mathrm{id}_2{=}\mathrm{not_Adidas}'{:}\mathit{Ppty}\\ \mathrm{id}_3{:}\left[\mathrm{e}{:}\mathrm{buy}(\Uparrow^3\mathrm{id}_1.\mathrm{x},\ \lambda P{:}\mathit{Ppty}\ .\ \left[\mathrm{e}{:}\mathrm{exist}(\Uparrow\mathrm{id}_1\wedge\Uparrow\mathrm{id}_2,\ P)\right])\right]\end{array}\right]{:}\mathit{RecType}\\
\mathrm{id}_2{:}\mathrm{want}^{\dagger}(\Uparrow^2\mathrm{id}_1.\mathrm{x},\ \mathrm{id}_1)\\
\mathrm{id}_3{=}\left[\mathrm{id}_1{=}\mathrm{jacket_like_Malte's}'{:}\mathit{Ppty}\right]{:}\mathit{RecType}\\
\mathrm{id}_4{:}\mathrm{pov}(\mathrm{id}_3,\ \mathrm{id}_1)
\end{array}\right]
\end{array}\right]$$

Adrian’s desire is perfectly rational given that he does not know that a jacket with three stripes on its sleeves is made by Adidas. Note that we can also truthfully report his desire even when we know about Adidas and the three stripes, as in (111).

(111) Adrian wants to buy a jacket like Malte’s but not from Adidas. He doesn’t realize that having three stripes on the sleeve means that the jacket is from Adidas.

It seems that none of these successive complications of the scenario, increasing, so to speak, the degree of intensionality involved, provide a problem when you combine an theory of intensional types with the notion of point of view as we have described it.

6.5 Summary of resources introduced

Items that are new since Chapter 5 are marked “**New!**” and items that have been revised since Chapter 5 are marked “**Revised!**”.

6.5.1 Universal grammar resources

6.5.1.1 Types

$$\mathit{Loc} \;\text{—}\; \begin{bmatrix} \text{x-coord} & : & \mathit{Real} \\ \text{y-coord} & : & \mathit{Real} \\ \text{z-coord} & : & \mathit{Real} \end{bmatrix}$$

Phon — a basic type

e : *Phon* iff e is a phonological event

$$\mathit{SEvent} \;\text{—}\; \begin{bmatrix} \text{e-loc} & : & \mathit{Loc} \\ \text{sp} & : & \mathit{Ind} \\ \text{au} & : & \mathit{Ind} \\ \text{e} & : & \mathit{Phon} \\ \text{c}_{\text{loc}} & : & \text{loc(e,e-loc)} \\ \text{c}_{\text{sp}} & : & \text{speaker(e,sp)} \\ \text{c}_{\text{au}} & : & \text{audience(e,au)} \end{bmatrix} \text{(as in Chapter 2)}$$

Assgnmnt — a basic type

r : *Assgnmnt* iff r : *Rec* and $\text{labels}(r) \subset \{\text{x}_0, \text{x}_1, \ldots\}$

PropCntxt — a basic type

r : *PropCntxt* iff r : *Rec* and $\text{labels}(r) \cap \{\text{x}_0, \text{x}_1, \ldots\} = \emptyset$

$$\mathit{Cntxt} \;\text{—}\; \begin{bmatrix} \mathfrak{s} & : & \mathit{Assgnmnt} \\ \mathfrak{c} & : & \mathit{PropCntxt} \end{bmatrix}$$

CntxtType — a basic type

T : *CntxtType* iff $T \sqsubseteq$ *Cntxt*

xType — a basic type

T : *xType* iff T : *RecType* and x $\in \text{labels}(T)$

$$\mathit{Ppty} \;\text{—}\; \begin{bmatrix} \text{bg} & : & \mathit{xType} \\ \text{fg} & : & (\text{bg}\rightarrow\mathit{RecType}) \end{bmatrix}$$

PlPpty — a basic type

P : *PlPpty* iff P : *Ppty* and for some type T, P.bg $\sqsubseteq \big[\text{x:plurality}(T)\big]$

$$\mathit{PPpty} \;\text{—}\; \begin{bmatrix} \text{bg} & : & \mathit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow\mathit{Ppty}) \end{bmatrix}$$

Quant — (*Ppty*→*RecType*)

$PQuant$ — $\begin{bmatrix} \text{bg} & : & \mathit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow \mathit{Quant}) \end{bmatrix}$

$QuantDet$ — $(\mathit{Ppty}\rightarrow \mathit{Quant})$

$PQuantDet$ — $\begin{bmatrix} \text{bg} & : & \mathit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow \mathit{QuantDet}) \end{bmatrix}$

$PRecType$ — $\begin{bmatrix} \text{bg} & : & \mathit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow \mathit{RecType}) \end{bmatrix}$

$Cont$ — $\mathit{PRecType}\vee\mathit{PPpty}\vee\mathit{PQuant}\vee\mathit{PQuantDet}$

Cat — a basic type

s, np, det, n, v, vp : Cat

Syn — $\begin{bmatrix} \text{cat} & : & \mathit{Cat} \\ \text{daughters} & : & \mathit{Sign}^* \end{bmatrix}$

$Sign$ — a basic type

$$\sigma : \mathit{Sign} \text{ iff } \sigma : \begin{bmatrix} \text{s-event} & : & \mathit{SEvent} \\ \text{syn} & : & \mathit{Syn} \\ \text{cont} & : & \mathit{Cont} \end{bmatrix}$$

$SignType$ — a basic type

$T : \mathit{SignType}$ iff $T \sqsubseteq \mathit{Sign}$

S — $\begin{bmatrix} \mathit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=s:}\mathit{Cat}\end{bmatrix} \end{bmatrix}$)

NP — $\begin{bmatrix} \mathit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=np:}\mathit{Cat}\end{bmatrix} \end{bmatrix}$

Det — $\begin{bmatrix} \mathit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=det:}\mathit{Cat}\end{bmatrix} \end{bmatrix}$

N — $\begin{bmatrix} \mathit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=n:}\mathit{Cat}\end{bmatrix} \end{bmatrix}$

V — $\begin{bmatrix} \mathit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=v:}\mathit{Cat}\end{bmatrix} \end{bmatrix}$

VP — $\begin{bmatrix} \mathit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=vp:}\mathit{Cat}\end{bmatrix} \end{bmatrix}$

$NoDaughters$ — $\begin{bmatrix}\text{syn:}\begin{bmatrix}\text{daughters=}\varepsilon\text{:}\mathit{Sign}^*\end{bmatrix}\end{bmatrix}$

Real — a basic type

n : *Real* iff n is a real number

Card — a basic type

n : *Card* iff n is a cardinal number (natural numbers with the addition of $\aleph_0, \aleph_1, \ldots$)

$$\textit{AmbTempFrame} \text{ — } \begin{bmatrix} \text{x} & : & \textit{Real} \\ \text{loc} & : & \textit{Loc} \\ \text{e} & : & \text{temp(loc, x)} \end{bmatrix}$$

$$\textit{TempRiseEventCntxt} \text{ — } \begin{bmatrix} \text{fix} & : & \begin{bmatrix} \text{loc} & : & \textit{Loc} \end{bmatrix} \\ \text{scale} & : & (\textit{AmbTempFrame} \to \textit{Real}) \end{bmatrix}$$

TempRiseEvent —

$$\lambda r{:}\textit{TempRiseEventCntxt} \;.\; \begin{bmatrix} \text{e} & : & (\textit{AmbTempFrame} \| r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale(e[0])} < r.\text{scale(e[1])} \end{bmatrix}$$

$$\textit{PriceFrame} \text{ — } \begin{bmatrix} \text{x} & : & \textit{Real} \\ \text{loc} & : & \textit{Loc} \\ \text{commodity} & : & \textit{Ind} \\ \text{e} & : & \text{price(commodity, loc, x)} \end{bmatrix}$$

$$\textit{PriceRiseEventCntxt} \text{ — } \begin{bmatrix} \text{fix} & : & \begin{bmatrix} \text{loc} & : & \textit{Loc} \\ \text{commodity} & : & \textit{Ind} \end{bmatrix} \\ \text{scale} & : & (\textit{PriceFrame} \to \textit{Real}) \end{bmatrix}$$

PriceRiseEvent —

$$\lambda r{:}\textit{TempRiseEventCntxt} \;.\; \begin{bmatrix} \text{e} & : & (\textit{PriceFrame} \| r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale(e[0])} < r.\text{scale(e[1])} \end{bmatrix}$$

$$\textit{LocFrame} \text{ — } \begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{loc} & : & \textit{Loc} \\ \text{e} & : & \text{at(x, loc)} \end{bmatrix}$$

$$\textit{LocRiseEventCntxt} \text{ — } \begin{bmatrix} \text{fix} & : & \begin{bmatrix} \text{x} & : & \textit{Ind} \end{bmatrix} \\ \text{scale} & : & (\textit{LocFrame} \to \textit{Real}) \end{bmatrix}$$

LocRiseEvent —

$$\lambda r{:}\textit{LocRiseEventCntxt} \;.\; \begin{bmatrix} \text{e} & : & (\textit{LocFrame} \| r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale(e[0])} < r.\text{scale(e[1])} \end{bmatrix}$$

Topos **New!** — a basic type

If τ : *Topos*, then $\tau : \begin{bmatrix} \text{bg} & : & \textit{Type} \\ \text{fg} & : & (\text{bg} \to \textit{Type}) \end{bmatrix}$

6.5.1.2 Predicates

with arity $\langle$*Phon*, *Loc*$\rangle$

loc — e : loc(u, l) iff u is located at l in e

with arity $\langle$*Phon*, *Ind*$\rangle$

speaker — e : speaker(u, a) iff u is the speaker of u in e

audience — e : audience(u, a) iff u is the audience of u in e

with arity $\langle$*Card*$\rangle$

card — X : card(n) iff for some T, $X : \mathrm{set}(T)$ and $|X| = n$

card_at_least — X : card_at_least(n) iff for some T, $X : \mathrm{set}(T)$ and $|X| \geq n$

card_at_most — X : card_at_most(n) iff for some T, $X : \mathrm{set}(T)$ and $|X| \leq n$

with arity $\langle$*Ppty*$\rangle$

unique — s : unique(P) iff $|\,[\downarrow P \restriction s]\,| = 1$

with arity $\langle$*Ppty*, *Ppty*$\rangle$

exist — s : exist(P,Q) iff $[\downarrow P] \cap [\downarrow Q \restriction s] \neq \emptyset$

every **New!** — s : every(P,Q) iff $[\downarrow P] \subseteq [\downarrow Q \restriction s]$

with arity $\langle$*PlPpty*, *PlPpty*$\rangle$

exactly_n **New!** — for n a natural number,
s : exactly_n(P, Q) iff s : at_least_n(P, Q)$\wedge$at_most_n(P, Q)

at_least_n **New!** — for n a natural number,
s : at_least_n(P, Q) iff $[\check{}\mathcal{F}((Q \restriction s).\mathrm{fg}\;|_{\mathcal{F}(P.\mathrm{fg})}) \mathbin{\dot{\wedge}} \big[\mathrm{x{:}card\_at\_least}(n)\big]\big] \neq \emptyset$

at_most_n **New!** — for n a natural number,
s : at_most_n(P, Q) iff $r : \mathcal{F}((Q \restriction s).\mathrm{fg}\;|_{\mathcal{F}(P.\mathrm{fg})})$ implies r : $\big[\mathrm{x{:}card\_at\_most}(n)\big]$

with arity $\{\langle T\rangle \mid T$ is a type$\}$

be — e : be(a) iff $a\varepsilon e$

with arity $\langle$*Loc*, *Real*$\rangle$

temp — e : temp(l, n) iff n is the temperature at l in e.

with arity $\langle \mathit{Real}, \mathit{Real} \rangle$

less-than — e : less-than(n, m) iff $n\varepsilon e$, $m\varepsilon e$ and $n < m$

with arity $\langle \mathit{Type}, \mathit{Type}, \mathit{Topos} \rangle$

nec **New!** —
If $\mathbb{T}$ is a modal type system and $p \in \mathbb{T}$, then
$s :_p \mathrm{nec}(T, B, \tau)$ iff $s :_p B$, $B \sqsubseteq_{\mathbb{T}} \tau.\mathrm{bg}$ and $\tau(s) \sqsubseteq_{\mathbb{T}} T$

poss **New!** —
If $\mathbb{T}$ is a modal type system and $p \in \mathbb{T}$, then
$s :_p \mathrm{poss}(T, B, \tau)$ iff $s :_p B$, $B \sqsubseteq_{\mathbb{T}} \tau.\mathrm{bg}$ and $\tau(s) \top_{\mathbb{T}} T$

with arity $\langle \mathit{RecType}, \mathit{RecType} \rangle$

pov **New!** — e : pov(T_1, T_2) iff T_2 is a point of view on T_1 in e.
e : pov(T_1, T_2) implies $\mathrm{labels}(T_2) \subseteq \mathrm{labels}(\mathrm{T}_1)$

with arity $\langle \mathit{Ind}, \mathit{RecType} \rangle$

ltm **New!** — e : ltm(a, T) iff T is a's long term memory in e.

rbelieve **New!** — e : rbelieve(a, T) iff T is a's religious beliefs in e.

des **New!** — e : des(a, T) iff T is a's desires in e.

6.5.1.3 Scales

(as in Chapter 5)

6.5.1.4 Lexicon

Lex

If T_{phon} is a phonological type (that is, $T_{\mathrm{phon}} \sqsubseteq$ *Phon*) and T_{sign} is a sign type (that is, $T_{\mathrm{sign}} \sqsubseteq$ *Sign*), then we shall use Lex(T_{phon}, T_{sign}) to represent

$$((T_{\mathrm{sign}} \mathbin{\dot\wedge} \left[\text{s-event:}\left[\text{e:}T_{\mathrm{phon}}\right]\right]) \mathbin{\dot\wedge} \mathit{NoDaughters})$$

SemCommonNoun(T_{bg}, p)

If p is a predicate with arity $\langle \mathit{Ind} \rangle$ and T_{bg} is a type (of context), then SemCommonNoun(T_{bg}, p) is

$$\ulcorner\lambda c{:}T_{\mathrm{bg}}\ .\ \ulcorner\lambda r{:}\left[\text{x:}\mathit{Ind}\right]\ .\ \left[\ \text{e} \quad : \quad p(r.\text{x})\ \right]\urcorner\urcorner$$

If p is a predicate with arity $\langle Rec\rangle$ and T_{bg} is a type (of context), then SemCommonNoun(T_{bg}, p) is

$$\ulcorner\lambda c{:}T_{\mathrm{bg}}\ .\ \ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Rec}\end{bmatrix}\ .\ \begin{bmatrix}\mathrm{e} & : & p(r.\mathrm{x})\end{bmatrix}\urcorner\urcorner$$

Lex$_{\mathrm{CommonNoun}}$(T_{phon}, T_{bg}, p)

If T_{phon} is a phonological type, p is a predicate with arity $\langle Ind\rangle$ or $\langle Rec\rangle$ and T_{bg} is a type (of context), then Lex$_{\mathrm{CommonNoun}}$(T_{phon}, T_{bg}, p) is

$$\mathrm{Lex}(T_{\mathrm{phon}}, \mathit{N})\ \dot{\wedge}\ \begin{bmatrix}\mathrm{cont{=}SemCommonNoun}(T_{\mathrm{bg}}, p){:}\mathit{PPpty}\end{bmatrix}$$

SemPropName(T_{phon})

If T_{phon} is a phonological type, then SemPropName(T_{phon}) is

$$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind} \\ \mathrm{e{:}named}(\mathrm{x}, T_{\mathrm{phon}})\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\mathit{Ppty}\ .\ P(c.\mathfrak{c})\urcorner$$

Lex$_{\mathrm{PropName}}$(T_{phon})

If T_{phon} is a phonological type,

then Lex$_{\mathrm{PropName}}$(T_{phon}) is

$$\mathrm{Lex}(T_{\mathrm{phon}}, \mathit{NP})\ \dot{\wedge}\ \begin{bmatrix}\mathrm{cnt{=}SemPropName}(T_{\mathrm{phon}}){:}\mathit{PQuant}\end{bmatrix}$$

SemPron

$$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt} \\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\mathit{Ppty}\ .\ P(\begin{bmatrix}\mathrm{x}{=}c.\mathfrak{s}.\mathrm{x}_0\end{bmatrix})\urcorner$$

LexPron(T_{phon})

If T_{phon} is a phonological type, then LexPron(T_{phon}) is

$$\mathrm{Lex}(T_{\mathrm{phon}}, \mathit{NP})\ \dot{\wedge}\ \begin{bmatrix}\mathrm{cont{=}SemPron}{:}\mathit{PQuant}\end{bmatrix}$$

SemNumeral(n)

If n is a real number, then SemNumeral(n) is

$$\ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda P{:}\mathit{Ppty}\ .\ P(\begin{bmatrix}\mathrm{x}{=}n\end{bmatrix})\urcorner$$

Lex$_{\mathrm{numeral}}$(T_{phon}, n) **New!**

If T_{phon} is a phonological type and n is a real number, then Lex$_{\mathrm{numeral}}$(T_{phon}, n) is

$$\mathrm{Lex}(T_{\mathrm{phon}}, \mathit{NP})\ \dot{\wedge}\ \begin{bmatrix}\mathrm{cnt{=}SemNumeral}(n){:}\mathit{PQuant}\end{bmatrix}$$

SemIndefArt

$$\lambda Q{:}\textit{Ppty}\ .\ \ulcorner\lambda c{:}\textit{Cntxt}\ .\ \lambda P{:}\textit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q & : & \textit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \textit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{bmatrix}\urcorner$$

$\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \textit{Det}) \mathbin{\dot{\wedge}} \big[\text{cont}{=}\text{SemIndefArt}{:}(\textit{Ppty}{\rightarrow}\textit{PQuant})\big]$$

SemUniversal

$$\lambda Q{:}\textit{Ppty}\ .\ \ulcorner\lambda c{:}\textit{Cntxt}\ .\ \lambda P{:}\textit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q & : & \textit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \textit{Ppty} \\ \text{e} & : & \text{every(restr, scope)} \end{bmatrix}\urcorner$$

$\text{Lex}_{\text{Universal}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{Universal}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \textit{Det}) \mathbin{\dot{\wedge}} \big[\text{cont}{=}\text{SemUniversal}{:}(\textit{Ppty}{\rightarrow}\textit{PQuant})\big]$$

SemDefArt

$$\lambda Q{:}\textit{Ppty}\ .\ \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt} \\ \mathfrak{c}{:}\big[\text{e:unique}(Q)\big]\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ \begin{bmatrix} \text{restr}{=}Q\restriction c.\mathfrak{c}.\text{e} & : & \textit{Ppty} \\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \textit{Ppty} \\ \text{e} & : & \text{every(restr, scope)} \end{bmatrix}\urcorner$$

$\text{Lex}_{\text{DefArt}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \textit{Det}) \mathbin{\dot{\wedge}} \big[\text{cont}{=}\text{SemDefArt}{:}(\textit{Ppty}{\rightarrow}\textit{PQuant})\big]$$

SemIntransVerb(T_{bg}, p)

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle\textit{Ind}\rangle$, then SemIntransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\big[\text{x}{:}\textit{Ind}\big]\ .\ \big[\ \text{e}\ :\ p(r.\text{x})\ \big]\urcorner\urcorner$$

If $T_{\text{bg}} \sqsubseteq \big[\mathfrak{c}{:}\mathit{Rec}\big]$ is a record type (for context) and p is a predicate with arity $\langle \mathit{Rec}, \mathit{Rec}\rangle$, then SemIntransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\big[\text{x}{:}\mathit{Rec}\big]\ .\ \big[\ \text{e}\ \ :\ \ p(r.\text{x},\ c.\mathfrak{c})\ \big]\urcorner\urcorner$$

$\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$

If T_{phon} is a phonological type, $T_{\text{bg}} \sqsubseteq \big[\mathfrak{c}{:}\mathit{Rec}\big]$ a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}\rangle$ or $\langle \mathit{Rec}, \mathit{Rec}\rangle$, then $\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$ is

$$\text{Lex}(T_{\text{phon}},\ V_i)\ \dot{\wedge}\ \big[\text{cnt=SemIntransVerb}(T_{\text{bg}},\ p){:}\mathit{PPpty}\big]$$

SemTransVerb(T_{bg}, p) **Revised!**

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}, \mathit{Ind}\rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \lambda\mathcal{Q}{:}\mathit{Quant}\ .\ \ulcorner\lambda r_1{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \mathcal{Q}(\ulcorner\lambda r_2{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \big[\ \text{e}\ \ :\ \ p(r_1.\text{x},\ r_2.\text{x})\ \big]\urcorner)\urcorner\urcorner$$

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}, \mathit{Quant}\rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \lambda\mathcal{Q}{:}\mathit{Quant}\ .\ \ulcorner\lambda r{:}\big[\text{x}{:}\mathit{Ind}\big]\ .\ \big[\ \text{e}\ \ :\ \ p(r.\text{x},\ \mathcal{Q})\ \big]\urcorner)\urcorner$$

$\text{Lex}_{\text{TransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$ **Revised!**

If T_{phon} is a phonological type, T_{bg} a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}, \mathit{Ind}\rangle$ or $\langle \mathit{Ind}, \mathit{Quant}\rangle$, then $\text{Lex}_{\text{TransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$ is

$$\text{Lex}(T_{\text{phon}},\ V_t)\ \dot{\wedge}\ \big[\text{cnt=SemTransVerb}(T_{\text{bg}},\ p){:}\mathit{PPpty}\big]$$

SemBe

SemBe_{ID}

$$\begin{aligned}&\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\big[\text{ty}{:}\mathit{Type}\big]\end{bmatrix}\ .\\ &\quad\lambda\mathcal{Q}{:}\mathit{Quant}\ .\\ &\qquad\ulcorner\lambda r_1{:}\big[\text{x}{:}c.\mathfrak{c}.\text{ty}\big]\ .\\ &\qquad\quad\mathcal{Q}(\ulcorner\lambda r_2{:}\big[\text{x}{:}c.\mathfrak{c}.\text{ty}\big]\ .\ \begin{bmatrix}\text{x=}r_1.\text{x},\ r_2.\text{x} & : & c.\mathfrak{c}.\text{ty}\\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner)\urcorner\urcorner\end{aligned}$$

$\text{SemBe}_{\text{scalar}}$

$$\begin{aligned}&\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{ty}{:}\mathit{Type}\\ \text{sc}{:}(\text{ty}\rightarrow\mathit{Real})\end{bmatrix}\end{bmatrix}\ .\\ &\quad\lambda\mathcal{Q}{:}\mathit{Quant}\ .\\ &\qquad\ulcorner\lambda r_1{:}\big[\text{x}{:}c.\mathfrak{c}.\text{ty}\big]\ .\\ &\qquad\quad\mathcal{Q}(\ulcorner\lambda r_2{:}\big[\text{x}{:}\mathit{Real}\big]\ .\ \begin{bmatrix}\text{x=}c.\mathfrak{c}.\text{sc}(r_1.\text{x}),\ r_2.\text{x} & : & \mathit{Real}\\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner)\urcorner\urcorner\end{aligned}$$

$\text{Lex}_{\text{be}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{be}_{\text{ID}}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \textit{V}) \mathbin{\dot{\wedge}} \left[\text{cont=SemBe}_{\text{ID}}\text{:}(\textit{Quant}\rightarrow\textit{Ppty})\right]$$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{be}_{\text{scalar}}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \textit{V}) \mathbin{\dot{\wedge}} \left[\text{cont=SemBe}_{\text{scalar}}\text{:}(\textit{Quant}\rightarrow\textit{Ppty})\right]$$

$\mathrm{FrameType}(p)$

$\mathrm{FrameType}$ is a partial function on predicates, p, with arity $\langle\textit{Ind}\rangle$ which can be defined for particular agents and particular times, which obeys the constraint:

$$\mathrm{FrameType}(p) \sqsubseteq \begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e} & : & p(\text{x}) \end{bmatrix}$$

p_frame **New!**

1. If p is a predicate in the domain of $\mathrm{FrameType}$, then p_frame is a predicate with arity $\langle\textit{Rec}\rangle$.
2. e : p_frame(r) iff r : $\mathrm{FrameType}(p)$ and $e = r$

p_pl

1. If p is a singular predicate (i.e. there is no p' such that $p = p'$_pl) with arity $\langle T\rangle$, then p_pl is a predicate with arity $\langle\mathrm{plurality}(T)\rangle$
2. e : p_pl(A) if for all $a \in A$, $e : p(a)$

CommonNounIndToFrame

If T_{phon} is a phonological type, p is a predicate with arity $\langle\textit{Ind}\rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then
CommonNounIndToFrame($\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$$\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p\text{\_frame})$$

RestrictCommonNoun

If T_{phon} is a phonological type, p is a predicate, T_{bg} and T_{res} are record types and Σ is $\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$, then RestrictCommonNoun(Σ, T_{res}) is

$$\Sigma \boxed{\dot{\wedge}} \left[\ \text{cont=}\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\text{SemCommonNoun}(T_{\text{bg}}, p)(c)\,|_{T_{\text{res}}}\urcorner\urcorner \quad : \quad \textit{PPpty}\ \right]$$

IntransVerbIndToFrame

If T_{phon} is a phonological type, p is a predicate with arity $\langle\textit{Ind}\rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then
IntransVerbIndToFrame($\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$$\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p\_\text{frame})$$

PluralCommonNoun

We assume that 'pluralnoun' is a function that maps phonological types for singular common nouns to corresponding phonological types for plural common nouns.

If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T\rangle$ and T_{bg} is a record type then PluralCommonNoun(Lex$_{\text{CommonNoun}}$(T_{phon}, T_{bg}, p)) =

$$\text{Lex}_{\text{CommonNoun}}(\text{pluralnoun}(T_{\text{phon}}), T_{\text{bg}}, p\_\text{pl})$$

PluralIntransVerb

We assume that 'pluralverb' is a function that maps phonological types for singular verbs to corresponding phonological types for plural verbs.

If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T\rangle$ and T_{bg} is a record type then PluralIntransVerb(Lex$_{\text{IntransVerb}}$(T_{phon}, T_{bg}, p)) =

$$\text{Lex}_{\text{IntransVerb}}(\text{pluralverb}(T_{\text{phon}}), T_{\text{bg}}, p\_\text{pl})$$

6.5.1.5 Constituent structure

(as in Chapter 5)

6.5.1.6 Action rules

LEXRES

$$\frac{\text{Lex}(T, C)\ \text{resource}_A \qquad u :_A T}{:_A\ (\text{Lex}(T, C) \underset{.}{\wedge} \left[\text{s-event:}\left[\text{e=}u\text{:}T\right]\right])}$$

TOPOSCONCLUDE **New!**

$$\frac{\tau : \textit{Topos} \qquad \tau\ \text{resource}_A \qquad s :_A \tau.\text{bg}}{:_A \tau(s)}$$

TOPOSPERMIT **New!**

$$\frac{\tau : \textit{Topos} \qquad \tau\ \text{resource}_A \qquad s :_A \tau.\text{bg}}{:_A \tau(s)!}$$

TOPOSOBLIGE **New!**

$$\frac{\tau : \textit{Topos} \qquad \tau\ \text{resource}_A \qquad s :_A \tau.\text{bg}}{:_A \tau(s)!}\ \text{oblig}$$

6.5.2 Universal speech act resources

(as in Chapter 2)

6.5.3 Universal dialogue resources

(as in Chapter 4)

6.5.4 English resources

6.5.4.1 Types and predicates

Basic phonological types for words

{"Dudamel", "is", "a", "conductor", "Beethoven", "composer", "Uchida", "pianist", "aha", "ok", "leaves" , "hugs", "dog", "nine", "ninety", "find" **New!**, "seek" **New!**, "worship" **New!**}

Predicates

with arity $\langle \mathit{Ind} \rangle$ {conductor, composer, pianist, leave, dog, passenger}

with arity $\langle \mathit{Ind}, \mathit{Quant} \rangle$ {hug **Revised!**, find **New!**, seek **New!**, worship **New!**, want$_Q$ **New!**}

e : worship(a,$\mathcal{Q}$) iff for some T

1. e : rbelieve(a, T)
2. $T \sqsubseteq_{\rightsquigarrow} \mathcal{Q}(\lambda r{:}\left[\text{x:}\mathit{Ind}\right] \,.\, \text{worship}^{\dagger}(a, r.\text{x}))$

or for some T'

1. $e :_{\underline{\varepsilon}}$ rbelieve(a, T)
2. $e :_{\underline{\varepsilon}}$ pov(T', T)
3. $T \boxed{\dot{\wedge}} T' \sqsubseteq_{\rightsquigarrow} Q(\lambda r{:}\left[\text{x:}\mathit{Ind}\right] \,.\, \text{worship}^{\dagger}(a, r.\text{x}))$

We restrict our attention to modal systems, $\mathbb{T}$, such that

1. if $p \in \{\text{hug}, \text{find}\}$ then
   $p(a, \mathcal{Q}) \approx_{\mathbb{T}} \mathcal{Q}(\ulcorner \lambda r{:}\left[\text{x:}\mathit{Ind}\right] \,.\, \left[\text{e:}p^{\dagger}(a, r.\text{x})\right] \urcorner)$
2. successful(seek(a, $\mathcal{Q}$)) $\sqsubseteq_{\mathbb{T}}$ find(a, $\mathcal{Q}$)
3. want$_Q$(a,Q) $\approx_{\mathbb{T}}$ want†(a, $Q(\ulcorner \lambda r{:}\left[\text{x:}\mathit{Ind}\right] \,.\, \text{have}(a, r.\text{x}) \urcorner)$)

with arity $\langle \mathit{Rec}, \mathit{Rec} \rangle$ — {rise }

e : rise(r, c) if

r : *AmbTempFrame*,
c : *TempRiseEventCntxt* and
e : *TempRiseEvent*(c) $\dot{\wedge}$ $\left[\text{e:}\left[\text{t}_0{=}r{:}\mathit{AmbTempFrame}\right]\right]$

or if

r : *PriceFrame*,
c : *PriceRiseEventCntxt* and
e : *PriceRiseEvent*(c) $\dot{\wedge}$ $\left[\text{e:}\left[\text{t}_0{=}r{:}\mathit{PriceFrame}\right]\right]$

or if

r : *LocFrame*,
c : *LocRiseEventCntxt* and
e : *LocRiseEvent*(c) $\dot{\wedge}$ $\left[\text{e}:\left[\text{t}_0{=}r{:}\textit{LocFrame}\right]\right]$

with arity $\langle \textit{Rec} \rangle$ — {temperature}

e : temperature(r) if

r : *AmbTempFrame* and $e = r$

with arity $\langle \textit{Ind}, \textit{TravelFrame} \rangle$ — {take_journey}

s : take_journey(a, e) iff $s = e$ and e.traveller = a

with arity $\langle \textit{Ind}, \textit{Ppty} \rangle$ — {want$_P$}

We restrict our attention to modal systems, $\mathbb{T}$, such that

want$_P$(a, P) $\approx_{\mathbb{T}}$ want†(a, P($\left[\text{x}{=}a\right]$))

with arity $\langle \textit{Ind}, \textit{RecType} \rangle$ — {believe **New!**, want† **New!**}

e : believe(a, T) if

e : ltm(a, T')
and $T' \sqsubseteq_{\rightsquigarrow} T$

or if

$e :_{\varepsilon}$ believe(a, T_1)
$e' :_{\varepsilon}$ pov(T_2, T_1)
and $T_1 \boxed{\dot{\wedge}} T_2 \sqsubseteq_{\rightsquigarrow} T$

e : want†(a, T) if for some T'
e : des(a, T')
and $T' \sqsubseteq_{\rightsquigarrow} T$

or if for some T_1 and T_2
$e :_{\varepsilon}$ want†(a, T_1)
$e :_{\varepsilon}$ pov(T_2, T_1)
and $T_1 \boxed{\dot{\wedge}} T_2 \sqsubseteq_{\rightsquigarrow} T$

Frame types

$$\textit{DogFrame} \text{ — } \begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \\ \text{age} & : & \textit{Real} \\ \text{c}_{\text{age}} & : & \text{age\_of(x,age)} \end{bmatrix}$$

$$\textit{TravelFrame} \text{ — } \begin{bmatrix} \text{traveller} & : & \textit{Ind} \\ \text{source} & : & \textit{Loc} \\ \text{goal} & : & \textit{Loc} \end{bmatrix}$$

$$\textit{PassengerFrame} \;—\; \begin{bmatrix} \mathrm{x} & : & \textit{Ind} \\ \mathrm{e} & : & \mathrm{passenger(x)} \\ \mathrm{journey} & : & \textit{TravelFrame} \\ \mathrm{c_{travel}} & : & \mathrm{take\_journey(x, journey)} \end{bmatrix}$$

6.5.4.2 Grammar

Lexical sign types

$\{$Lex$_{\text{PropName}}$(“Dudamel”),
Lex$_{\text{PropName}}$(“Beethoven”),
Lex$_{\text{Pron}}$(“he”),
Lex$_{\text{numeral}}$(“nine”, 9),
Lex$_{\text{numeral}}$(“ninety”, 90),
Lex$_{\text{IndefArt}}$(“a”),
Lex$_{\text{Universal}}$(“every”),
Lex$_{\text{DefArt}}$(“the”),
Lex$_{\text{CommonNoun}}$(“composer”, *Rec*, composer),
Lex$_{\text{CommonNoun}}$(“conductor”, *Rec*, conductor),
Lex$_{\text{CommonNoun}}$(“dog”, *Rec*, dog) (= $\Sigma_{\text{“dog”}}$),
RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“dog”}}$), *DogFrame*),
Lex$_{\text{CommonNoun}}$(“passenger”, *Rec*, passenger) (= $\Sigma_{\text{“passenger”}}$),
RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“passenger”}}$), *PassengerFrame*),
Lex$_{\text{CommonNoun}}$(“temperature”, *Rec*, temperature) (= $\Sigma_{\text{“temperature”}}$),
RestrictCommonNoun($\Sigma_{\text{“temperature”}}$, *AmbTempFrame*),
Lex$_{\text{IntransVerb}}$(“leave”, *Rec*, leave),
Lex$_{\text{IntransVerb}}$(“run”, *Rec*, run),
Lex$_{\text{IntransVerb}}$(“rise”, $\begin{bmatrix}\mathfrak{c}{:}\textit{TempRiseEventCntxt}\end{bmatrix}$, rise),
Lex$_{\text{IntransVerb}}$(“rise”, $\begin{bmatrix}\mathfrak{c}{:}\textit{PriceRiseEventCntxt}\end{bmatrix}$, rise),
Lex$_{\text{IntransVerb}}$(“rise”, $\begin{bmatrix}\mathfrak{c}{:}\textit{LocRiseEventCntxt}\end{bmatrix}$, rise),
Lex$_{\text{TransVerb}}$(“hug”, *Rec*, hug),
Lex$_{\text{TransVerb}}$(“find”, *Rec*, find) **New!**,
Lex$_{\text{TransVerb}}$(“seek”, *Rec*, seek) **New!**,
Lex$_{\text{TransVerb}}$(“worship”, *Rec*, worship) **New!**,
Lex$_{\text{be}_{\text{ID}}}$(“is”),
Lex$_{\text{be}_{\text{scalar}}}$(“is”),
Lex(“ok”, *S*),
Lex(“aha”, *S*) $\}$

Constituent structure rule components

CnstrIsA

$$\lambda u{:}V \mathbin{\dot{\wedge}} \begin{bmatrix}\text{s-event:}\begin{bmatrix}\text{e:“is”}\end{bmatrix}\end{bmatrix}{}^\frown NP \mathbin{\dot{\wedge}} \begin{bmatrix}\text{syn:}\begin{bmatrix}\text{daughters:}Det \mathbin{\dot{\wedge}} \begin{bmatrix}\text{s-event:}\begin{bmatrix}\text{e:“a”}\end{bmatrix}\end{bmatrix} \\ {}^\frown N \mathbin{\dot{\wedge}} \begin{bmatrix}\text{cont:}Ppty\end{bmatrix}\end{bmatrix}\end{bmatrix}.$$
$$VP \mathbin{\dot{\wedge}} \begin{bmatrix}\text{cont=}u[2]\text{.syn.daughters}[2]\text{.cont:}Ppty\end{bmatrix}$$

Constituent structure rules

$$\begin{aligned}&\{S \longrightarrow NP\ VP \mid NP'(VP'{:}Ppty){:}RecType,\\ &\ NP \longrightarrow Det\ N \mid Det'({}_{@@}N'{:}PPpty){:}PQuant,\\ &\ VP \longrightarrow V\ NP \mathbin{\dot{\wedge}} \text{CnstrIsA},\\ &\ VP \longrightarrow V\ NP \mid V'(NP'{:}Quant){:}Ppty\}\end{aligned}$$

6.6 Summary

In this chapter we first pointed out some conceptual and technical problems involving possible worlds as they are standardly used in semantics. We have considered the two main areas where possible worlds have been used: modality and intensionality involving the attitudes. We have suggested that both benefit from an analysis in terms of intensional types instead of possible worlds.

Chapter 7

Witness-based quantification

7.1 Introduction

In this chapter we are going to explore how the TTR approach to semantics we have developed gives us a novel treatment of quantification and binding in natural language.

In Section 7.2 we are going to propose a revision of witness conditions for quantificational ptypes which will constitute what we will call a *witness based* account of generalized quantifiers. The motivation for this is in part to get a neater treatment of anaphora, and a more general treatment along the lines of Lücking and Ginzburg (2019).

In Section 7.5 we will take a brief look at how long distance dependencies can be treated as preparation for our treatment of quantifier scope and binding in Chapter 8.

7.2 Quantifiers and their witness sets

7.2.1 Conservativity and dynamic generalized quantifiers

Here we review the treatment of quantifiers that we have presented so far and its relation to the notion of *conservativity* which we will discuss here. This property of conservativity facilitates the witness-based account of quantifiers that we will undertake below.

In Chapter 5, example (100), we introduced the notion of dynamic generalized quantifier and pointed out that one of the original motivations for them was what is known as donkey anaphora which we will discuss in Section 8.3. Here we will point out a connection between dynamic quantifiers and conservativity of quantifiers, noted in Chierchia (1995). The informal way to state conservativity for quantifiers is as in (1a) and an example is given in (1b).

(1) a. Q A B is true just in case Q A A&B is true

b. *every farmer likes a donkey* is true just in case *every farmer is a farmer and likes a donkey* (or more naturally, *every farmer is a farmer who likes a donkey*) is true

Most, if not all, natural language quantifiers have this property.[1]

Now consider the discussion of dynamic generalized quantification in Chapter 5, Section 5.5. There in example (100) we gave dynamic versions of generalized quantifier interpretations of noun-phrases. We spell out the witness condition for the quantificational ptype, (2b), corresponding to *every dog runs* in (2).

(2) a. $\text{dog}' = \ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix}\ .\ \begin{bmatrix}\text{e:dog}(r.\text{x})\end{bmatrix}\urcorner$

$\text{run}' = \ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix}\ .\ \begin{bmatrix}\text{e:run}(r.\text{x})\end{bmatrix}\urcorner$

b. $T = \text{every}(\text{dog}', \text{run}' \mid_{\mathfrak{F}(\text{dog}')})$

c. $s : T$ iff $[{\downarrow}\text{dog}'] \subseteq [{\downarrow}\text{run}'|_{\mathcal{F}(\text{dog}')} \restriction s]$ (witness condition for 'every', p. 238)

iff $[{\downarrow}\text{dog}'] \subseteq [{\downarrow}\text{run}'|_{\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:dog(x)}\end{bmatrix}} \restriction s]$ (fixed point types, p. 244)

iff $[{\downarrow}\text{dog}'] \subseteq [{\downarrow}\ \lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix} \dot{\wedge} \begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:dog(x)}\end{bmatrix}\ .\ \begin{bmatrix}\text{e:run}(r.\text{x})\end{bmatrix} \restriction s\ ]$ (function restriction, p. 245)

iff $[{\downarrow}\text{dog}'] \subseteq [{\downarrow}\ \lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:dog(x)}\end{bmatrix}\ .\ \begin{bmatrix}\text{e:run}(r.\text{x})\end{bmatrix} \restriction s\ ]$ (merge, p. 88)

iff $[{\downarrow}\text{dog}'] \subseteq [{\downarrow}\ \lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:dog(x)}\end{bmatrix}\ .\ \begin{bmatrix}\text{e}\underline{\varepsilon}s\text{:run}(r.\text{x})\end{bmatrix}\ ]$ (property restriction, p. 127)

iff $\{a \mid \exists r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix} \wedge r.\text{x} = a \wedge [\check{}\begin{bmatrix}\text{e:dog}(r.\text{x})\end{bmatrix}] \neq \emptyset\}$

$\subseteq \{a \mid \exists r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{e:dog(x)}\end{bmatrix} \wedge r.\text{x} = a \wedge [\check{}\begin{bmatrix}\text{e}\underline{\varepsilon}s\text{:run}(r.\text{x})\end{bmatrix}] \neq \emptyset\}$ (${\downarrow}$, p. 233)

iff $\{a \mid [\check{}\text{dog}(a)] \neq \emptyset\} \subseteq \{a \mid [\check{}\text{dog}(a)] \neq \emptyset \wedge \exists s'[s'\underline{\varepsilon}s \wedge s' : \text{run}(a)]\}$ (Arity of 'dog', 'run' and set extension of records, p. 130ff)

iff $\{a \mid \exists s'[s' : \text{dog}(a)]\} \subseteq \{a \mid \exists s'[s' : \text{dog}(a)] \wedge \exists s'[s'\underline{\varepsilon}s \wedge s' : \text{run}(a)]\}$ ($[\check{}T]$, p. 125)

7.2.2 Witness sets

The classical view of quantifiers is based on the notion that noun phrases represent sets of sets or set of properties and the definition of a quantifier involves characterizing which set of sets or properties it represents. This was the view presented, for example, in Barwise and Cooper (1981). Associated with this was the notion of *witness set* defined by Barwise and Cooper as in (3).

[1]For discussion, see Peters and Westerståhl (2006), p. 138f.

(3) A *witness set* for a quantifier $D(A)$ living on A is any subset w of A such that $w \in D(A)$.

In (3) D was used as the function corresponding to a determiner such as *some* or *most* mapping a set A (corresponding, for example, to the set denoted by a common noun phrase such as *farmer* or *farmer who owns a donkey*) to a set of sets. The notion *lives on* used by Barwise and Cooper corresponds to what was later in the literature referred to as *conservativity*. Their definition of the lives-on property, slightly simplified by removing reference to the model, is given in (4).

(4) A quantifier Q *lives on* a set A if Q is a set of sets with the property that

$$X \in Q \text{ iff } (X \cap A) \in Q$$

This means that the notion of witness set given by Barwise and Cooper is defined for conservative quantifiers. Examples of witness sets that they give include: a witness set for the quantifier corresponding to a proper name *John* as the singleton set containing the individual John; a witness set corresponding to *a woman* as any non-empty set of women; a witness set corresponding to *most women* as a set of women which contains most women.

The notion of witness set was introduced by Barwise and Cooper in a section called *Processing Quantified Statements*. It was introduced as an auxiliary notion which could be used in an account of how an agent might evaluate the truth of a quantified statement. This suggests that it should play an important role in a theory of semantics like ours which is oriented towards explaining cognitive semantic processing, especially if it is a theory which attempts to do this in terms of judgements that objects (including situations) are witnesses of types. It seems natural to make a link between the notion of witnesses for types and the notion of witnesses for quantifiers. We will go further and suggest that the characterization of the meaning of determiners is based on witness sets, thus elevating the witness sets from an auxiliary notion derived from the meaning assigned to quantifiers to the central notion which characterizes the distinctions between the various quantifier meanings available, just as in type theory the notion of meaning is characterized in terms of the witness conditions for types. In doing this we will be going at least part way to meeting some of the requirements of the proposals by Ginzburg and Purver (2008); Lücking and Ginzburg (2019).

In the literature on generalized quantifiers van Benthem (1984) introduced the perspective that we should think of determiners as representing relations between sets rather than as mappings from sets to families of sets. This is reflected in our characterization of quantifier relations as relations between properties (which can be used to generate the set of objects which have the property) and the use of ptypes constructed with quantifier relations and two properties as arguments.

With each quantifier relation, q, and property, P, we will associate a type of witness sets $q^w(P)$. For example, a set, X, is of type $\text{most}^w(P)$ if X is a set of objects with property P which contains most of the objects which have property P. We will say that a witness for the quantificational ptype 'most(P, Q)' is a pair (coded as a record and thus corresponding intuitively to a situation) consisting of the set, X, where $X : q^w(P)$ (i.e. X is a witness set for q and P) and a function, f, whose domain is X and such that for any $a \in X$, $f(a)$ is a situation which shows that a has property Q. In general for distributive readings of monotone increasing quantifiers, q, we can say that a witness for $q(P, Q)$ provides a witness set X of type $q^w(P)$ and a function which shows that every member of X has the property Q. For distributive readings of monotone decreasing quantifiers we need a different kind of function together with the witness set. Here we have to check that everything which has both property P and property Q is a member of the witness set. Thus we need a function, f, whose domain is the set of objects having both P and Q, such that if a is in this set then $f(a)$ is a situation which shows that a is a member of the witness set X. These two kinds of functions correspond exactly to the evaluation procedures suggested in Barwise and Cooper (1981) quoted in (5).

(5) *To evaluate $X \in D(A)$ do the following:*

1. Take some subset w of A which you know to be in $D(A)$
2. (i) For mon↑ $D(A)$, check $w \subseteq X$.
   (ii) For mon↓ $D(A)$, check $(X \cap A) \subseteq w$
3. If there is such a w, the sentence is true. Otherwise it is false.

Using pairs of witness sets and functions as witnesses for quantificational ptypes is also closely related to the treatment of quantification in Martin-Löf type theory using Σ-types and dependent types. (See, for example, discussion in Ranta, 1994.) A witness for the Σ-type (6a) would be an ordered pair as characterized in (6b).

(6) a. $(\Sigma x : A)B(x)$
    b. $\langle a, b\rangle$ where $a : A$ and $b : B(a)$

In the following subsections we will develop the tools we need to make our analysis precise in terms of the TTR machinery we have developed so far to create a witness-based analysis of quantfiers. We will also consider how we can move away from a set-based account of quantification to a type-based approach where we can estimate the probability of a quantificational ptype being witnessed on the basis of our previous experience.

7.2.3 Relating properties, types and sets

A property, P, can be *pure*, as in (7a), that is, P.bg has exactly one field, the required 'x'-field; or it can be restricted by including additional fields in the background as in (7b), the restricted property of being an individual which is a dog that barks.

(7) a. $\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix} . \begin{bmatrix}\text{e} & : & \text{bark}(r.\text{x})\end{bmatrix}\urcorner$

b. $\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{c:dog(x)}\end{bmatrix} . \begin{bmatrix}\text{e} & : & \text{bark}(r.\text{x})\end{bmatrix}\urcorner$

The ability to restrict properties will be important for analyzing dynamic quantification where information from the first argument of the quantifier relation is passed to the second argument of the quantifier relation and this is what enables the treatment of donkey anaphora. Thus *every farmer who owns a donkey likes it* will be treated as the 'every'-relation holding between the property of being a farmer and owning a donkey and the property of being a farmer who owns a donkey and likes it thus providing an antecedent for *it* within the second property (see Section 8.3). However, it will also be important to be able to "purify" such restricted properties, that is, relate them systematically to a corresponding property whose background contains just the one 'x'-field. This will allow us to avoid the proportion problem that can arise in the analysis of donkey anaphora, that is in computing whether a sentence like *most farmers who own a donkey like it* is true, we need to ensure that the majority of farmers who own a donkey are such that they like it and not that the majority of pairs of farmers and donkeys where the farmer owns the donkey are such that the farmer likes the donkey. Thus we need a property of individuals, not of farmer-donkey pairs. Suppose we have the restricted property (8a). We will define an operation on functions which will yield the pure property (8b).

(8) a. $\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{c}_1\text{:farmer(x)}\\ \text{y:}\mathit{Ind}\\ \text{c}_2\text{:donkey(y)}\\ \text{e:own(x, y)}\end{bmatrix} . \begin{bmatrix}\text{e} & : & \text{like}(r.\text{x}, r.\text{y})\end{bmatrix}\urcorner$

b. $\ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix} . \begin{bmatrix}\mathfrak{c} & : & \begin{bmatrix}\text{x}{=}r.\text{x:}\mathit{Ind}\\ \text{c}_1\text{:farmer(x)}\\ \text{y:}\mathit{Ind}\\ \text{c}_2\text{:donkey(y)}\\ \text{e:own(x, y)}\end{bmatrix}\\ \text{e} & : & \begin{bmatrix}\text{e} & : & \text{like}(\Uparrow\mathfrak{c}.\text{x}, \Uparrow\mathfrak{c}.\text{y})\end{bmatrix}\end{bmatrix}\urcorner$

This purification operation changes the property in (8a), a property of farmers who own a donkey into the property (8b), a property of individuals. The restriction in (8a) has been lowered into the body of the property and labelled with '$\mathfrak{c}$', intuitively a local context in the type returned by the function.

We first define an operation which will generalize a type to the type containing one of its non-dependent fields. This is just one way of many to operate on a type to make it more general. Consider (9).

(9) $\left[\begin{array}{lll} \text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \end{array}\right]$

We can pick out the 'x'-field which is not dependent, unlike the 'e'-field. The result is (10)

(10) $\left[\begin{array}{lll} \text{x} & : & \textit{Ind} \end{array}\right]$

In general for a record type, T, we will represent the generalization of T to its non-dependent ℓ-field as T^{ℓ}.

The general definition of this is (11), repeated in Appendix 16.

(11) If T : *RecType*, $\ell \in$ labels(T) and $\langle \ell, T' \rangle \in T$ where T': *Type* (that is, $[\ell, T']$ is a non-dependent field in T), then *the generalization of* T *to its* ℓ*-field*, T^{ℓ}, is

$$\left[\begin{array}{lll} \ell & : & T' \end{array}\right]$$

We denote the purification operation on properties by $\mathfrak{P}$ and define it as in (12).

(12) If P : *Ppty*, then

if P.bg^{x} = P.bg, then

$\mathfrak{P}(P) = P$

otherwise:

$\mathfrak{P}(P)$ is $\ulcorner\lambda r{:}P.\text{bg}^{\text{x}}\ .\ \left[\begin{array}{lll} \mathfrak{c} & : & P.\text{bg} \parallel \left[\text{x}{=}r.\text{x}\right] \\ \text{e} & : & P(\mathfrak{c}) \end{array}\right]\urcorner$

(Recall that $T \parallel r$ is the result of specifying or anchoring T with the record r as defined in Appendix 15.)

(12) represents one of two options for purifying a property. It yields the property of being something such that the background conditions P.bg are met *and* the body of the property is met under those conditions. The alternative is to say that the property yielded is the property of being something such that *if* the background conditions P.bg are met, *then* the body of the property is met under those conditions. This alternative, which has the effect of universal quantification over situations which meet the background conditions, will we represented by the operator '$\mathfrak{P}^{\forall}$' characterized in (13).

(13) If P : *Ppty*, then

if P.bg^{x} = P.bg, then

$\mathfrak{P}^{\forall}(P) = P$

otherwise:

$\mathfrak{P}^{\forall}(P)$ is $\ulcorner\lambda r{:}P.\text{bg}^{\text{x}}\ .\ ((r'{:}P.\text{bg}\,\|\,[\text{x}{=}r.\text{x}]) \rightarrow \left[\ \text{e}\ :\ P(r')\ \right])\urcorner$

The option presented by '$\mathfrak{P}^{\forall}$' will become relevant in our discussion of donkey anaphora on p. 436. For now, we will continue the discussion with the existential variant, '$\mathfrak{P}$'.

As an example let us apply $\mathfrak{P}$ to (8a). The result will be (14)

$$(14)\quad \ulcorner\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right]\ .\ \left[\begin{array}{lll}\mathfrak{c} & : & \left[\begin{array}{l}\text{x}{:}\mathit{Ind}\\ \text{c}_1{:}\text{farmer(x)}\\ \text{y}{:}\mathit{Ind}\\ \text{c}_2{:}\text{donkey(y)}\\ \text{e}{:}\text{own(x, y)}\end{array}\right] \,\|\, \left[\text{x}{=}r.\text{x}\right]\\ \text{e} & : & \ulcorner\lambda r{:}\left[\begin{array}{l}\text{x}{:}\mathit{Ind}\\ \text{c}_1{:}\text{farmer(x)}\\ \text{y}{:}\mathit{Ind}\\ \text{c}_2{:}\text{donkey(y)}\\ \text{e}{:}\text{own(x, y)}\end{array}\right]\ .\ \left[\ \text{e}\ :\ \text{like}(r.\text{x}, r.\text{y})\ \right]\urcorner(\mathfrak{c})\end{array}\right]\urcorner$$

According to the definition of the specification of a record type by a record, (15a) represents (15b).

$$(15)\quad \text{a.}\quad \left[\begin{array}{l}\text{x}{:}\mathit{Ind}\\ \text{c}_1{:}\text{farmer(x)}\\ \text{y}{:}\mathit{Ind}\\ \text{c}_2{:}\text{donkey(y)}\\ \text{e}{:}\text{own(x, y)}\end{array}\right] \,\|\, \left[\text{x}{=}r.\text{x}\right]$$

$$\text{b.}\quad \left[\begin{array}{l}\text{x}{=}r.\text{x}{:}\mathit{Ind}\\ \text{c}_1{:}\text{farmer(x)}\\ \text{y}{:}\mathit{Ind}\\ \text{c}_2{:}\text{donkey(y)}\\ \text{e}{:}\text{own(x, y)}\end{array}\right]$$

To understand the reduction of the type in the 'e'-field in (14) we represent it first in the official notation for dependent fields as in (16a). This represents the same as (16b) (by β-conversion) and its abbreviatory notation in the context of (14) is (16c).

(16) a. $\langle \lambda v{:}\begin{bmatrix}\text{x:}\textit{Ind}\\ \text{c}_1\text{:farmer(x)}\\ \text{y:}\textit{Ind}\\ \text{c}_2\text{:donkey(y)}\\ \text{e:own(x,y)}\end{bmatrix} . \ulcorner\lambda r{:}\begin{bmatrix}\text{x:}\textit{Ind}\\ \text{c}_1\text{:farmer(x)}\\ \text{y:}\textit{Ind}\\ \text{c}_2\text{:donkey(y)}\\ \text{e:own(x,y)}\end{bmatrix} . \begin{bmatrix}\text{e} : \text{like}(r.\text{x},r.\text{y})\end{bmatrix}\urcorner(v), \langle\mathfrak{c}\rangle\rangle$

b. $\langle \lambda v{:}\begin{bmatrix}\text{x:}\textit{Ind}\\ \text{c}_1\text{:farmer(x)}\\ \text{y:}\textit{Ind}\\ \text{c}_2\text{:donkey(y)}\\ \text{e:own(x,y)}\end{bmatrix} . \begin{bmatrix}\text{e} : \text{like}(v.\text{x},v.\text{y})\end{bmatrix}, \langle\mathfrak{c}\rangle\rangle$

c. $\begin{bmatrix}\text{e} : \text{like}(\Uparrow\mathfrak{c}.\text{x},\Uparrow\mathfrak{c}.\text{y})\end{bmatrix}$

Making these substitutions yields (8b).

If P is a pure property, we will use the notation $P\{a\}$ to represent the type $P(\begin{bmatrix}\text{x}=a\end{bmatrix})$. If $P\{a\}$ is witnessed we say that a has property P. We can now define the type of objects which have P, which we will represent as $\mathfrak{T}(P)$. We introduce this type as in (17).

(17) a. If P : *Ppty* and P is pure, then $\mathfrak{T}(P)$: *Type*.

b. $a : \mathfrak{T}(P)$ iff $\mathfrak{P}(P)\{a\}$ is witnessed.

There is a different route to a type with the same witnesses as $\mathfrak{T}(P)$. We have previously defined $[\downarrow P]$ as the set of objects which have the property P, as in Chapter 5, p. 233. For any set, X, we can define a type whose witnesses are exactly the members of X. We will represent this type as $\mathfrak{T}(X)$ and introduce it as in (18) (repeated in Appendix 5.

(18) a. If T is a type and $X : \mathrm{set}(T)$, then $\mathfrak{T}(X)$ is a type

b. $a : \mathfrak{T}(X)$ iff $a \in X$

Given this, it is straightforward to see that (19) holds.

(19) For any property, P, $a : \mathfrak{T}(P)$ iff $a : \mathfrak{T}([\downarrow P])$

What distinguishes these two types is the method they suggest for determining whether something is a witness for the type. In the case of $\mathfrak{T}([\downarrow P])$ we have to first determine the complete set of objects which have the property and then determine whether the object in question is a member of the set. In the case of $\mathfrak{T}(P)$ we only have to determine whether the object in question has the property P. Computing the set of all objects which have a property may be viable when we are considering a property whose extension is a small finite set of objects (for example, if the property is that of being a dog in a particular small situation) but it does not seem feasible in the case of large finite sets or infinite sets. We will return to this issue below when we consider the interpretation of generalized quantifiers which are classically treated by comparing sets and we will consider an alternative in terms of types and estimated probabilities.

7.2.4 Types of witness sets for quantifiers

In this section we will discuss the characterization of types of witness sets for various generalized quantifiers as a step on our way to characterizing a witnessed-based account of generalized quantifiers. The witness sets we characterize will be very close to those of Barwise and Cooper (1981), though not exactly the same in all cases. In general for a quantifier relation, q, and property, P, a witness set, X, of type $q^w(P)$ must meet two conditions. The first is that it must be a subset of the property extension of P. We now have two ways of expressing this as shown in (20).

(20) a. $X \subseteq [{\downarrow}P]$
  b. $X : \mathrm{set}(\mathfrak{T}(P))$

(20a) makes explicit the connection to the original definition of witness sets by Barwise and Cooper. (20b) is an equivalent condition on X which does not involve the computation of the complete property extension of P.

The second condition which must be met by witnesses, X, of $q^w(P)$ is a cardinality condition on X. This may be an absolute condition on the size of X or it may involve a comparison of the size of X with the size of the property extension of P, that is, $[{\downarrow}P]$. Thus even though we have a way of avoiding the computation of the total property extension in the first condition on witness sets, we will not always be able to avoid it in the second condition. It is for this reason that we will move to probability estimations in the next section.

The witness condition for ‘exist$^{\mathrm{w}}(P)$’ is given in (21).

(21) $X : \mathrm{exist}^{\mathrm{w}}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $|X| = 1$

Note that this differs from Barwise and Cooper in that it requires that the witness set contain exactly one object having property P rather than at least one such object. The quantifier relation 'exist' is used in interpreting the English determiner *a* and also the singular determiner *some*. Plural *some* corresponds to the quantifier relation 'exist$_{\text{pl}}$'. The witness condition for 'exist$^{\text{w}}_{\text{pl}}(P)$' is given in (22).

(22) X : exist$^{\text{w}}_{\text{pl}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \geq 2$

Correspondingly we can define the witness condition for 'no$^{\text{w}}(P)$' is given in (23).

(23) X : no$^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| = 0$

(23) could, of course, be given more concisely as (24).

(24) X : no$^{\text{w}}(P)$ iff $X = \emptyset$

The witness condition for 'every$^{\text{w}}(P)$' is given in (25).

(25) X : every$^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| = |[\check{\mathfrak{T}}(P)]|$

Note that (25) requires X to be identical with $[\check{\mathfrak{T}}(P)]$ and we could, of course, express the witness condition more succinctly as (26) if we are not concerned about demonstrating that the witness conditions for the types witness sets for all quantifiers follow the same pattern.

(26) X : every$^{\text{w}}(P)$ iff $X = [\check{\mathfrak{T}}(P)]$

Either way, we seem committed to computing the set $[\downarrow P]$ in order to compute a witness set of 'every' and P and this can therefore lead to problems if $[\downarrow P]$ is a large set. A standard way of avoiding the computation of this set in the interpretation of universal quantification is to associate universal quantification with a function so that, using the notions we have built up, a witness for 'every(P,Q)' would be a function from $\mathfrak{T}(P)$ to $\mathfrak{T}(Q)$ (see Ranta, 1994, for discussion of how this is done in a standard Martin-Löf type theory). This avoids computing the property extension of P if we have the right view of functions as intensional objects or procedures, rather than the von Neumann notion of function as a set of ordered pairs as is standard in set theory. We will make use of such functions when we come to treat the witness conditions for quantificational ptypes. Such a treatment on its own does not yield a characterization of a witness set, however, and thus does not immediately yield a way of treating plural discourse anaphora with a universal quantifier as antecedent as in (27).

(27) Every dog ran into the field. They had seen the rabbits.

For this reason we will pursue the witness set approach and attempt to solve the problem of large sets by introducing probability estimation.

In order to account for the witness condition for 'most$^{\text{w}}$' we assume that there is a threshold, $\theta_{\text{most}}(P)$, which tells you what proportion of the property extension of P has to be included in the witness set. This is possibly an oversimplification in that the threshold may depend on more than the quantifier relation and the first argument property to the relation. A common assumption in the generalized quantifier literature is that 'most(P,Q)' is true just in case at least one more than half of the Ps are Q (see, for example, Peters and Westerståhl, 2006). This may be true if the property extension of P is a small finite set. But it hardly seems to be the case for an example involving a larger set as in (28).

(28) Most supporters in the stadium cheered when the goal was scored.

(28) does not appear to be true if only one more than half of the forty thousand supporters in the stadium cheered. Rather we would expect the number of cheering supporters to be something in excess of 75% or 85% of the supporters in the stadium. The unclarity as to exactly which proportion is involved leads us to introduce a threshold which can vary with the context and the speaker.

The witness condition for 'most$^{\text{w}}(P)$' is given in (29).

(29) X : most$^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\text{most}}(P)$

Clause 2 in (29) requires us to count both the witness set, which may be quite large, like the number of supporters in a stadium, as well as the property extension of P. We will address this by using probability estimations in the next section.

The English determiner *many*, has two readings: absolute and proportional. We will treat this in terms of two quantifier relations 'many$_a$' and 'many$_p$'. For any property, P, we will assume that thresholds, $\theta_{\text{many}_a}(P)$ and $\theta_{\text{many}_p}(P)$ are provided. These will indicate, respectively, the number of objects having property P that will count as many and the proportion of the set of objects having P that will count as many. The witness condition for '$\text{many}_a^{\text{w}}(P)$' is given in (30).

(30) $X : \text{many}_a^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \geq \theta_{\text{many}_a}(P)$

The witness condition for '$\text{many}_p^{\text{w}}(P)$' is given in (31).

(31) $X : \text{many}_p^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\text{many}_p}(P)$

The quantifier relations corresponding to *few* are treated in an exactly similar fashion to those corresponding to *many* except that the cardinality of the witness set or the proportion of the property extension included in the witness set is required to be less than or equal to the relevant threshold. The witness condition for '$\text{few}_a^{\text{w}}(P)$' is given in (32).

(32) $X : \text{few}_a^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \leq \theta_{\text{few}_a}(P)$

The witness condition for '$\text{few}_p^{\text{w}}(P)$' is given in (33).

(33) $X : \text{few}_p^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\downarrow P]|} \leq \theta_{\text{few}_p}(P)$

The quantifier relations corresponding to *a few* use the same thresholds as those corresponding to *few* but in the case of *a few* the size of the witness set and the proportion of the witness set to the property extension have to be greater than or equal to the threshold. The witness condition for ‘$\text{a\_few}^{\text{w}}_{a}(P)$’ is given in (34).

(34) $X : \text{a\_few}^{\text{w}}_{a}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \geq \theta_{\text{few}_a}(P)$

The witness condition for ‘$\text{a\_few}^{\text{w}}_{p}(P)$’ is given in (35).

(35) $X : \text{a\_few}^{\text{w}}_{p}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\text{few}_p}(P)$

7.3 Relating witness sets to probabilities

In general our strategy for relating witness sets to probabilities will involve two conditions, the first of which is the same as we had in the previous section, that is that the witness set, X, is a set of objects which have the property, P, i.e. $X : \text{set}(\mathfrak{T}(P))$. The second condition, however, will place a constraint on the value of a conditional probability which we will represent as $p(\mathfrak{T}(X)\|\mathfrak{T}(P))$, that its, the probability for any object a that it is of type $\mathfrak{T}(X)$ given that it is of type $\mathfrak{T}(P)$.[2] Similar probabilities associated with quantifiers have been suggested by Emerson (2020). Here we will take a frequentist view of this probability and define it according the equation in (36).

(36) $p(T_1\|T_2) = \frac{|[\check{\ }T_1 \wedge T_2]|}{|[\check{\ }T_2]|}$ if T_2 is witnessed and 0 otherwise.

Clearly, this of itself will not help if we wish to avoid counting the set of witnesses of T_1 or T_2. However, probabilities can be estimated on the basis of previous experience. We will assume that an agent has available in memory a finite set of Austinian propositions, $\mathfrak{J}$, recording judgements previously made. We will define a notion of having a type with respect to a set of Austinian propositions, $\mathfrak{J}$, using $a :_{\mathfrak{J}} T$ to represent “a is of type T with respect to $\mathfrak{J}$”. This notion is defined in (37).

[2]We contrast this with $p(\mathfrak{T}(X)|\mathfrak{T}(P)$, the probability that there is something of type $\mathfrak{T}(X)$, given that there is something of type $\mathfrak{T}(P)$. See Cooper *et al.* (2014a) for discussion.

(37) a. $a :_{\mathfrak{J}} T$ if $\begin{bmatrix} \text{sit} & = & a \\ \text{type} & = & T \end{bmatrix} \in \mathfrak{J}$

b. If $T = (T_1 \wedge T_2)$, then $a :_{\mathfrak{J}} T$ if $a :_{\mathfrak{J}} T_1$ and $a :_{\mathfrak{J}} T_2$

c. Otherwise $a \not:_{\mathfrak{J}} T$

We will use the notation $[\check{T}]_{\mathfrak{J}}$ to represent the extension of T with respect to $\mathfrak{J}$, defined in (38).

(38) $[\check{T}]_{\mathfrak{J}} = \{a \mid a :_{\mathfrak{J}} T\}$

We can now define the notion of *estimate of* $p(T_1 \| T_2)$ *based on* $\mathfrak{J}$, $p_{\mathfrak{J}}(T_1 \| T_2)$, as in (39).

(39) $p_{\mathfrak{J}}(T_1 \| T_2) = \frac{|[\check{T_1} \wedge T_2]_{\mathfrak{J}}|}{|[\check{T_2}]_{\mathfrak{J}}|}$

A measure of reliability of the estimate could be related to the number of instances observed, that is, about which a judgement has been made, for example as in (40).

(40) reliability($p_{\mathfrak{J}}(T_1 \| T_2)$) = ln min($|[\check{T_1}]_{\mathfrak{J}}|, |[\check{T_2}]_{\mathfrak{J}}|$)

This could still involve an agent in a serious amount of counting which might be unintuitive from a psychological point of view. From a computational point of view it would be straightforward enough to keep track of how many objects of each type have already been judged and to increment these numbers when a new object of the type is encountered. However, we do not seem to be aware of how many objects of a given type we have encountered when the numbers get high. For example, I know that I have seen a lot of dogs in my life but I have no idea how many. It would also not explain how I could estimate the probability that any person in a stadium is wearing an IFK Göteborg scarf just by looking around the stadium but not exactly counting the number of people in the stadium and the number of those wearing the scarf. This seems to point to the related proposals based on Austinian propositions involving Bayesian reasoning about probability which are suggested in Cooper *et al.* (2014a). An important difference between what we are doing here and what we did in the earlier work is that here we assume that the Austinian propositions in $\mathfrak{J}$ are categorical rather than probabilistic. We could, of course, derive a set of categorical propositions from a set of probabilistic propositions by choosing categorical propositions for all those in the probabilistic set whose probabilities exceed a given threshold.

Here we will look at the straight frequentist interpretation of probabilities associated with types of witness sets of quantifiers as this can be shown to relate directly to the characterization of these types in Section 7.2.4 and assume that these probabilities can be estimated on the basis of a (tractably small) set, $\mathfrak{J}$, of propositions available to the agent in memory. For convenience in the discussion below we will repeat the second clause of the non-deterministic characterization from Section 7.2.4 for comparison.

The witness condition for ‘exist$^{\mathrm{w}}(P)$’ is given in (41).

(41) X : exist$^{\mathrm{w}}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) = \frac{1}{|[\check{\ }\mathfrak{T}(P)]|}$ (corresponds to $|X| = 1$)

It is easy to see that the clauses (1.) and (2.) in (41) are equivalent to the non-probabilistic version if we take the frequentist interpretation of the conditional probability in (36). Nothing is gained by going to the extra expense of computing the probability here. All we need to do is check that the witness set is a singleton and that its member is of the type $\mathfrak{T}(P)$.

The witness condition for ‘exist$^{\mathrm{w}}_{\mathrm{pl}}(P)$’ is given in (42).

(42) X : exist$^{\mathrm{w}}_{\mathrm{pl}}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \frac{2}{|[\check{\ }\mathfrak{T}(P)]|}$ (corresponds to $|X| \geq 2$)

The probabilistic condition is again equivalent to the non-probabilistic condition and there is no point in going to the extra expense of computing the probability.

The witness condition for ‘no$^{\mathrm{w}}(P)$’ is given in (43).

(43) X : no$^{\mathrm{w}}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) = 0$ (corresponds to $|X| = 0$)

Again the probabilistic and non-probabilistic conditions are equivalent and there is no point to computing the probability since all we have to do is check that the witness set is the empty set.

The witness condition for ‘every$^{\mathrm{w}}(P)$’ is given in (44).

(44) X : every$^{\mathrm{w}}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) = 1$ (corresponds to $|X| = |[\check{\ }\mathfrak{T}(P)]|$)

Again the probabilistic and non-probabilistic conditions are equivalent given the frequentist interpretation of probability. However, here there is some point to the probability for semantic processing. Both the conditions as formulated require counting the set of objects which have the property, P, which can be intractable especially if the set is infinite or finite and reasonably large. However, the probability can be estimated on the basis of a finite number of observations. The more relevant observations you have, the more reliable your estimate. Consider the sentence in (45).

(45) Every dog barks when it is time to go for a walk

An utterance of (45) is naturally interpreted to be quantifying over dogs in general, or at least those which are physically capable of barking. I have no practical way of determining the truth or falsity of this sentence, though I can make an estimate on the basis of my observations of dogs in pre-walk situations. If all relevant observations of dogs involved the dog barking, then I can estimate that the sentence is true. Of course, if I have only observed two or three dogs, my estimate is not very reliable even though it is consistent with my experience. If on the other hand I have observed hundreds of situations where a dog is about to go for a walk, all of them with the dog barking, then it seems like a more reliable estimate, although my experience will not show conclusively that it will be true. How do sentences like this get used in a dialogue. Consider the (constructed) dialogue in (46).

(46) *A and B are about to take A's dog for a walk. The dog, realizing that a walk is in the offing, begins to bark excitedly.*
A : I must apologize for the racket.
B : Not to worry. Every dog barks when it is time to go for a walk. *B is thinking of her past experience of dogs in similar situations.*
A : Yes, that's right. *A is thinking of* her *past experience of dogs in similar situations.*

A and B are agreeing on the basis of their own distinct observations. They both know what the sentence means but they both also know that it is not practically possible for a human agent to verify the truth of the sentence on the set-based view or its equivalent frequentist probabilistic interpretation and therefore that the basis for the assertion must be a probability estimation. The situation seems similar to that with predicates of personal taste on the kind of approach taken by Cooper (2015, 2017a). Consider the (constructed) dialogue in (47).

(47) *A and B are eating lunch together and have just been served soup*
A : *(tasting the soup)* Hhm, this soup is good. *A is basing her assertion on her taste sensations.*
B : *(also tasting the soup)* You're right. It is. *B is basing her assertion on* her *taste sensations.*

Again A and B are agreeing on the basis of their own distinct observations. The reason here, however, does not have to do with the impracticality of counting a large set but rather that it is not possible to directly observe another person's taste sensation. But there is also an important difference between (46) and (47). In (46) there is a fact of the matter which is being discussed. For example, consider the continuation of (46) given in (48) where a third dialogue participant, C, joins the conversation.

(48) C : Actually, I used to have a dog which never barked except when he saw another dog or a squirrel.
B : OK, then, *most* dogs bark when it's time for a walk / #Well, *I* think that every dog barks when it's time for a walk

This contrasts with a continuation of (47) in a similar vein.

(49) C : No, this soup is not good.
A : #OK, then the soup is sort of/mostly good / Well, *I* think it's good.

What the examples have in common is that the justification for the assertions of the same "proposition" is different facts based on personal experience. Where they differ is in whether there is an objective fact or not.

A similar discussion holds for *most*. The witness condition for '$\text{most}^{\text{w}}(P)$' is given in (50).

(50) $X : \text{most}^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \theta_{\text{most}}(P)$ (corresponds to $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\text{most}}(P)$)

As usual the frequentist interpretation of the conditional probability is equivalent to the corresponding non-probabilistic. In order to see this note that (51) holds.

(51) $\frac{|[\check{\ }\mathfrak{T}(X)\wedge\mathfrak{T}(P)]|}{|[\check{\ }\mathfrak{T}(P)]|} = \frac{|X|}{|[\downarrow P]|}$

(51) holds because of the equalities in (52).

(52) a. $|[\check{\mathfrak{T}}(X)]| = |X|$ since $[\check{\mathfrak{T}}(X)] = X$
b. $[\check{\mathfrak{T}}(P)] = [\downarrow P]$
c. $|[\check{\mathfrak{T}}(X) \wedge \mathfrak{T}(P)]| = |[\check{\mathfrak{T}}(X)]|$ since $[\check{\mathfrak{T}}(X)] \subseteq [\check{\mathfrak{T}}(P)]$ (since $X : \text{set}(\mathfrak{T}(P))$)

Again, to make an evaluation when large sets are involved we may need to estimate the relevant probability on the basis of our own experience and the kind of dialogue which we illustrated for *every* might occur.

The witness condition for '$\text{many}^{\text{w}}_{a}(P)$' is given in (53).

(53) $X : \text{many}^{\text{w}}_{a}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P)$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \frac{\theta_{\text{many}_a}(P)}{|[\check{\mathfrak{T}}(P)]|}$ (corresponds to $|X| \geq \theta_{\text{many}_a}(P)$)

Again the probabilistic condition on the frequentist interpretation is equivalent to the non-probabilistic one since both require that the cardinality of X is greater than or equal to $\theta_{\text{many}_a}(P)$.

The witness condition for '$\text{many}^{\text{w}}_{p}(P)$' is given in (54).

(54) $X : \text{many}^{\text{w}}_{p}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \theta_{\text{many}_p}(P)$ (corresponds to $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\text{many}_p}(P)$)

Again, the frequentist interpretation of the probabilisitic condition is equivalent to the non-probabilistic condition and with large sets we may need to estimate the probability rather than compute the cardinality of the witness set.

The same holds for the quantifier relations corresponding to *few* and *a few*. The witness condition for '$\text{few}^{\text{w}}_{a}(P)$' is given in (55).

(55) $X : \text{few}^{\text{w}}_{a}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$

2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \leq \frac{\theta_{\text{few}_a}(P)}{[[\check{~}\mathfrak{T}(P)]]}$ (corresponds to $|X| \leq \theta_{\text{few}_a}(P)$)

The witness condition for ‘$\text{few}^{\text{w}}_{p}(P)$’ is given in (56).

(56) X : $\text{few}^{\text{w}}_{p}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \leq \theta_{\text{few}_p}(P)$ (corresponds to $\frac{|X|}{|[\downarrow P]|} \leq \theta_{\text{few}_p}(P)$)

The witness condition for ‘$\text{a\_few}^{\text{w}}_{a}(P)$’ is given in (57).

(57) X : $\text{a\_few}^{\text{w}}_{a}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \frac{\theta_{\text{few}_a}(P)}{[[\check{~}\mathfrak{T}(P)]]}$ (corresponds to $|X| \geq \theta_{\text{few}_a}(P)$)

The witness condition for ‘$\text{a\_few}^{\text{w}}_{p}(P)$’ is given in (58).

(58) X : $\text{a\_few}^{\text{w}}_{p}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \theta_{\text{few}_p}(P)$ (corresponds to $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\text{few}_p}(P)$)

7.4 Witness conditions for quantificational ptypes

In general there are two witness conditions that can be associated with quantificational ptypes $q(P, Q)$ where q is a quantifier relation and P and Q are properties. These correspond to the two evaluation procedures suggested by Barwise and Cooper (1981) in connection with witness sets for monotone increasing and decreasing quantifiers respectively. The two conditions are given in (59).

(59) a. $s : q(P, Q)$ iff $s : \begin{bmatrix} \text{X} & : & q^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{bmatrix}$

b. $s : q(P, Q)$ iff $s : \begin{bmatrix} \text{X} & : & q^w(P) \\ \text{f} & : & ((a : (\mathfrak{T}(P) \wedge \mathfrak{T}(Q))) \rightarrow \begin{bmatrix} \text{x=}a & : & \mathfrak{T}(\text{X}) \end{bmatrix}) \end{bmatrix}$

(59a) is the condition to be associated with monotone increasing quantifiers. It says that s is of the quantificational ptype just in case it provides an appropriate witness set (in a field labelled 'X') and a function (in a field labelled 'f') from objects, a, in that witness set to situations (modelled as records) which show that a has the purified property derived from second argument of the quantificational ptype. (59b) is the condition to be associated with monotone decreasing quantifiers. It says that s is of the quantificational ptype just in case it provides an appropriate witness set (in a field labelled 'X') and a function from objects, a, which have both the property which is the first argument and the purified property derived from the second argument to a situation (record) which shows that a is a member of the witness set.

For each quantifier relation, q, we have to say whether ptypes constructed with q have the witness condition (59a) or (59b). We shall call these the *general* witness conditions. Judging from the anaphoric possibilities associated with natural language quantified expressions the witness conditions used for some quantifier relations are not the general witness conditions but simpler conditions (which we will call *particular* witness conditions) using an equivalent type. That is, if the relevant witness condition expressed in (59) is $s : T$ what actually gets used is $s : T'$ where T is witnessed if and only if T' is witnessed. We shall discuss these cases as we go through the witness conditions associated with the individual quantifier relations.

The general witness condition for 'exist(P,Q)' is given in (60).

(60) $s : \text{exist}(P, Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \text{exist}^{w}(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{array}\right]$

This requires that a witness for 'exist(P,Q)' must be a record providing a pair of a singleton set whose member is an object which has property P and a function whose domain is this set which returns for any a a situation in which a has (the purification of) the property Q. Thus if we let 'dog′' and 'bark′' represent the properties indicated in (61a) and (61b) respectively then a witness for 'exist(dog′,bark′)' will according to (60) be of the type (61c).

(61) a. $\ulcorner\lambda r{:}\left[\text{x}{:}\textit{Ind}\right] . \left[\begin{array}{lll} \text{e} & : & \text{dog}(r.\text{x}) \end{array}\right]\urcorner$

b. $\ulcorner\lambda r{:}\left[\text{x}{:}\textit{Ind}\right] . \left[\begin{array}{lll} \text{e} & : & \text{bark}(r.\text{x}) \end{array}\right]\urcorner$

c. $\left[\begin{array}{lll} \text{X} & : & \text{exist}^{w}(\text{dog}') \\ \text{f} & : & ((a : \mathfrak{T}(X)) \rightarrow \text{bark}'\{a\}) \end{array}\right]$

It is trivial to show that (61c) has a witness just in case (62) has a witness.

(62) $\left[\begin{array}{lll} \text{x} & : & \mathfrak{T}(\text{dog}') \\ \text{e} & : & \text{bark}'\{\text{x}\} \end{array}\right]$

The argument is essentially that there is a singleton set containing a dog all of whose members bark just in case there is a dog which barks. We might prefer to have the particular witness condition for 'exist(P,Q)' in (63).

(63) $s : \text{exist}(P, Q)$ iff $s : \left[\begin{array}{lll} \text{x} & : & \mathfrak{T}(P) \\ \text{e} & : & \mathfrak{P}(Q)\{\text{x}\} \end{array}\right]$

Apart from its intuitive simplicity and correspondence to the classical DRT treatment of indefinites as well as to the use of Σ-types to interpret indefinites in type theory the particular witness condition provides a component in the witness (in the 'x'-field) which can be picked up on by singular anaphora in examples like (64).

(64) A dog is barking. It is right outside my window

The general witness condition for 'exist_{pl}(P,Q)' is (65).

(65) $s : \text{exist}_{\text{pl}}(P, Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \text{exist}^{w}_{\text{pl}}(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{array}\right]$

This corresponds to a distributive reading of plural *some*. It says intuitively that a situation, s, is of type 'exist_{pl}(P,Q) just in case s provides a set of objects which have property P with at least two members and a function which shows that each member of the set has property Q. In this case the witness set provides us with a suitable antecedent for plural anaphora as in (66) and we do not need a particular witness condition.

(66) Some dogs are barking. They are right outside my window.

The general witness condition for 'no(P,Q)' is given in (67).

(67) $s : \text{no}(P, Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \text{no}^{w}(P) \\ \text{f} & : & ((a : (\mathfrak{T}(P) \wedge \mathfrak{T}(Q))) \rightarrow \left[\begin{array}{lll} \text{x}{=}a & : & \mathfrak{T}(\text{X}) \end{array}\right]) \end{array}\right]$

The only witness set allowed (that is, the only set of type $\text{no}^{w}(P)$, no matter what P is) is the empty set. Suppose that we find some object, a, which has both properties P and Q, then the function will return a situation (record) that shows that a is in the empty set. There is, however, no such situation. The only way that there can be a function of this type is if the set of objects which have both P and Q is also the empty set. While this is technically correct and fits the general pattern for monotone decreasing quantifiers it does not seem to be an intuitive account of how we would check that 'no(P,Q)' is witnessed. More intuitive is to check each object that has property P and find that it does not have property Q. This method is equivalent to the first and the relationship between the two corresponds to the equivalence between (68a) and (68b) in first order logic.

(68) a. $\neg\exists x[P(x) \wedge Q(x)]$
b. $\forall x[P(x) \rightarrow \neg Q(x)]$

In order to do this we will use the notion of negation given in Cooper and Ginzburg (2011a, 2012). First we say that two types, T_1 and T_2 *preclude each other*, $T_1 \perp T_2$, just in case there is no possibility in which $[\check{T}_1]$ and $[\check{T}_2]$ overlap. That is, nothing can be of both types. We then introduce negated types as in (69).

(69) a. if T is a type, then $\neg T$ is a type
b. $a : \neg T$ iff for some type, T', such that $T \perp T'$, $a : T'$

We will define the particular witness condition for 'no(P,Q)' as (70).

(70) $s : \text{no}(P, Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \text{every}^w(P) \\ \text{f} & : & ((x : \mathfrak{X}(X)) \rightarrow \neg\mathfrak{P}(Q)\{x\}) \end{array}\right]$

Some evidence that English uses the particular witness condition rather than the general one comes from the fact that we can have plural discourse anaphora related to a noun phrase with *no* as its determiner, as in (71).

(71) No dog barked. They were all busy gnawing on a bone.

Clearly, *they* does not refer to the witness set of type 'no^w(dog$'$)', which would have to be the empty set, but rather to a witness set of type 'every^w(dog$'$)', the set of all dogs. This is an instance of *complement set anaphora*, first discussed in the psychology literature by Moxey and Sanford (1987); Sanford and Moxey (1993) and discussed in the semantics literature by Kibble (1997); Nouwen (2003); Lücking and Ginzburg (2019) among others.

The general witness condition for 'every(P,Q)' is given in (72).

(72) $s : \text{every}(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{every}^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{bmatrix}$

In this case we do not need a particular witness condition. (72) correctly predicts the availability of plural discourse anaphora as in (73).

(73) Every dog barked. They had been disturbed by the intruder.

The same holds for *most*. The general witness condition for 'most(P,Q)' is given in (74).

(74) $s : \text{most}(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{most}^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{bmatrix}$

In this case we do not need a particular witness condition. (74) correctly predicts the availability of plural discourse anaphora as in (75).

(75) Most dogs bark when somebody unknown comes into their territory. They are disturbed by an intruder.

Note that the occurrence of *they* here can be interpreted to refer not to all dogs or dogs in general but to the witness set of dogs containing most dogs. (This is what Moxey and Sanford call REFSET anaphora.) It can, however, also be interpreted to refer to dogs in general, that is, all dogs are disturbed by an intruder but not all of them bark when this happens. This is referred to as MAXSET anaphora by Moxey and Sanford. This could be taken as motivation for having a field which introduces the property 'dog' in the sign corresponding to *most dogs* in the manner suggested in Chapter 3. Alternatively, it might be considered as motivation for an additional field corresponding to P in the witness for 'most(P,Q)'. We will leave this issue unresolved. It seems clear, though, as has been pointed out in the literature (Nouwen, 2003), that COMPSET anaphora is not possible with *most*. That is, *they* in (76) cannot refer to dogs which do not bark when somebody unknown comes into their territory.

(76) #Most dogs bark when somebody unknown comes into their territory. They never feel threatened whatever happens.

The lack of this reading is consistent with the witness condition in (78).

Similar remarks can be made for *many*. The general witness condition for 'many$_a$(P,Q)' is given in (77).

(77) $s : \text{many}_a(P, Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \text{many}^w_a(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{array}\right]$

The general witness condition for 'many$_p$(P,Q)' is given in (78).

(78) $s : \text{many}_p(P, Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \text{many}^w_p(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{array}\right]$

For both absolute and proportional readings of *many* REFSET and MAXSET anaphora are available but not COMPSET. There is no motivation for a particular witness condition.

The general witness condition for 'few$_a$(P,Q)' is given in (79).

(79) $s : \text{few}_a(P, Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \text{few}^w_a(P) \\ \text{f} & : & ((a : (\mathfrak{T}(P) \wedge \mathfrak{T}(Q))) \rightarrow \left[\begin{array}{lll} \text{x}{=}a & : & \mathfrak{T}(\text{X}) \end{array}\right]) \end{array}\right]$

The general witness condition for 'few$_p$(P,Q)' is exactly similar as given in (80).

(80) $s : \text{few}_p(P, Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \text{few}^w_p(P) \\ \text{f} & : & ((a : (\mathfrak{T}(P) \wedge \mathfrak{T}(Q))) \rightarrow \left[\begin{array}{lll} \text{x}{=}a & : & \mathfrak{T}(\text{X}) \end{array}\right]) \end{array}\right]$

These witness conditions involve computing first the set of all objects that have both property P and Q and then checking that all the members of the set are in the witness set. A more intuitive and less computationally expensive way of achieving an equivalent result is to find a sufficient number of objects that have property P but which don't have property Q (in our terms they have a property which precludes them having Q). What counts as a sufficient number of objects? It has to be a number of objects with property P such that only few objects having property P remain, that is, it has to be a set with cardinality at least the cardinality of the set whose members have P minus the threshold, θ_{few} in the case of an absolute quantifier. In the case of a proportional quantifier the proportion of objects having P but not Q has to be greater than one minus the proportional threshold. We treat this by introducing types of complement witness sets for *few*. The absolute case is given in (81).

(81) a. If $\text{few}^w_a(P)$ is a type, then $\overline{\text{few}^w_a(P)}$ is a type

b. $X : \overline{\text{few}^w_a(P)}$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \geq |[\mathfrak{T}(P)]| - \theta_{\text{few}_a}(P)$

The proportional case is given in (82).

(82) a. If $\text{few}_p^w(P)$ is a type, then $\overline{\text{few}_p^w(P)}$ is a type

b. $X : \overline{\text{few}_p^w(P)}$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\check{\ }\mathfrak{T}(P)]|} \geq 1 - \theta_{\text{few}_p}(P)$

Clearly, computing whether a set is a witness for one of these new types can involve computing the cardinality of the set of objects which have property P and for this reason is may be more tractable to estimate a probability. The relevant probabilities here are similar to those associated with *many*. The witness condition for $\overline{\text{few}_a^w(P)}$ is given in (83).

(83) $X : \overline{\text{few}_a^w(P)}$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \frac{\theta_{\text{few}_a}(P)}{[[\check{\ }\mathfrak{T}(P)]]}$

The witness condition for $\overline{\text{few}_p^w(P)}$ is given in (84).

(84) $X : \overline{\text{few}_p^w(P)}$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $p(\mathfrak{T}(X)\|\mathfrak{T}(P) \geq 1 - \theta_{\text{few}_p}(P)$

Now we can give particular witness conditions for ptypes constructed with the quantifier predicates 'few_a' and 'few_p'. The witness condition for '$\text{few}_a(P,Q)$' is given in (85).

(85) $s : \text{few}_a(P,Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \overline{\text{few}_a^w(P)} \\ \text{f} & : & ((x : \mathfrak{T}(X)) \rightarrow \neg\mathfrak{P}(Q)\{x\}) \end{array}\right]$

The witness condition for '$\text{few}_p(P,Q)$' is given in (86).

(86) $s : \text{few}_p(P,Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \overline{\text{few}_p^w(P)} \\ \text{f} & : & ((x : \mathfrak{T}(X)) \rightarrow \neg\mathfrak{P}(Q)\{x\}) \end{array}\right]$

Why do we use complement witness set types for *few* in the particular witness conditions rather than witness sets related to *many*? We might naively have thought that *few dogs barked* was equivalent to *many dogs did not bark*. A standard analysis of *few* in the literature is as *not many*, that is *few dogs barked* is equivalent to *not many dogs barked* or *it is not true that many dogs barked*. On the kind of analysis that we are proposing here neither of these will work. We illustrate this with an example. Suppose that we have a dog hotel with twenty-five dogs in residence. Suppose that what counts as many dogs in the context is ten and what counts as few is five. This means that few dogs barked means that five or less dogs barked. If this is true then it follows that many dogs did not bark (fifteen or more, and we only need ten to count as many) and it also follows that it is not the case that many dogs barked (since at most five did and for many dogs to bark we would need ten). Thus *few* seems to imply both *many not* and *not many*. However, the implications do not go back the other way. Suppose that many dogs did not bark. This means that ten or more dogs did not bark. If ten dogs did not bark then fifteen did bark. Thus many dogs barked at the same time as many dogs did not bark. It does not follow that few dogs barked. Suppose that it is not the case that many dogs barked. This means that less than ten dogs barked. For example, nine dogs barked. While nine is not many in this context neither is it few. We might say *quite a lot* or *quite a few*. That is, our analysis allows for a gap between what counts as many and what counts as few and thus *not many* is not equivalent to *few*.

Adopting the particular witness condition predicts the existence of COMPSET anaphora as in (87).

(87) Few dogs in the kennels barked. They didn't hear the intruder.

However, there are convincing examples in the literature that *few* will also allow REFSET anaphora, which would be predicted by the general witness condition. Nouwen (2003) quotes (88) from Evans (1980).

(88) Few congressmen admire Kennedy, and they are very junior.

One way to handle this is to allow both the general and particular witness conditions as alternatives. Another solution to consider is letting the general witness condition be used for absolute *few* and the particular witness condition for proportional *few*. This would perhaps be consistent with Kibble's (1997) observation that complement anaphora seems to be associated with monotone decreasing proportional quantifiers.

The general witness condition for '$\text{a\_few}_a(P,Q)$' is given in (89).

(89) $s : \text{a\_few}_a(P, Q)$ iff $s : \left[\begin{array}{lll} \text{X} & : & \text{a\_few}^w_a(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{array}\right]$

The general witness condition for ‘a_few$_p$(P,Q)’ is given in (90).

(90) s : a_few$_p(P, Q)$ iff s : $\begin{bmatrix} \text{X} & : & \text{a\_few}_p^w(P) \\ \text{f} & : & ((a : \mathfrak{I}(\text{X})) \to \mathfrak{P}(Q)\{a\}) \end{bmatrix}$

These will correctly predict the availability of REFSET anaphora as in (91).

(91) A few dogs barked. They had heard the intruder.

If these are the only witness conditions then this will correctly predict the unavailability of COMPSET anaphora as shown by (92), where *they* cannot refer to the dogs that did not bark.

(92) #A few dogs barked. They hadn’t heard the intruder

7.4.1 Some examples

In this section we show how these definitions could be used to express content for English utterances, ignoring for the moment the parametric properties using in Chapter 5 and expressing the content of a quantified declarative sentence as a quantificational ptype rather than a record type containing a quantificational ptype as in Chapter 3 and subsequent chapters.

A dog barks The content of the indefinite article, *a(n)*, is given in (93). This is ‘SemIndefArt’

(93) $\lambda Q{:}\mathit{Ppty}\ .$
$\ulcorner \lambda c{:}\mathit{Cntxt}\ .$
$\lambda P{:}\mathit{Ppty}\ .$
$\begin{bmatrix} \text{restr=}Q & : & \mathit{Ppty} \\ \text{scope=}P\ |_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{bmatrix}\urcorner$

The content of *a dog* would then be as in (94).

(94) $\ulcorner \lambda c{:}\begin{bmatrix} \mathit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix} \text{f:}\mathit{PropCntxt} \\ \text{a:}\mathit{PropCntxt} \end{bmatrix} \end{bmatrix}\ .$
$\lambda P{:}\mathit{Ppty}\ .$
$\begin{bmatrix} \text{restr=dog}' & : & \mathit{Ppty} \\ \text{scope=}P\ |_{\mathfrak{F}(\text{dog}')} & : & \mathit{Ppty} \\ \text{e} & : & \text{exist(restr, scope)} \end{bmatrix}\urcorner$

Finally, the content of *a dog barks* would be as in (95).

(95) $$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{f}{:}\textit{PropCntxt}\\ \text{a}{:}\begin{bmatrix}\text{f}{:}\textit{PropCntxt}\\ \text{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix}.\begin{bmatrix}\text{restr=dog}' & : & \textit{Ppty}\\ \text{scope=bark}'\,|_{\mathfrak{F}(\text{dog}')} & : & \textit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{bmatrix}\urcorner$$

Following the particular witness condition for ‘exist(P,Q)’ in (63) we can infer (96).

(96) s : exist(dog′,bark′$|_{\mathcal{F}(dog')}$) iff

$$s : \begin{bmatrix}\text{x} & : & \mathfrak{T}(\text{dog}')\\ \text{e} & : & \mathfrak{P}(\text{bark}'|_{\mathcal{F}(\text{dog}')})\{\text{x}\}\end{bmatrix}$$

$\mathcal{F}$(dog′) is given in (97).

(97) $$\begin{bmatrix}\text{x} & : & \textit{Ind}\\ \text{e} & : & \text{dog(x)}\end{bmatrix}$$

‘bark′$|_{\mathcal{F}(\text{dog}')}$’ is thus (98).

(98) $$\lambda r{:}\begin{bmatrix}\text{x}{:}\textit{Ind}\\ \text{e}{:}\text{dog(x)}\end{bmatrix}.\begin{bmatrix}\text{e} & : & \text{bark}(r.\text{x})\end{bmatrix}$$

This means that ‘$\mathfrak{P}$(bark′$|_{\mathcal{F}(\text{dog}')}$)’ is (99).

(99) $$\lambda r{:}\begin{bmatrix}\text{x}{:}\textit{Ind}\end{bmatrix}.\begin{bmatrix}\mathfrak{c} & : & \begin{bmatrix}\text{x}=r.\text{x} & : & \textit{Ind}\\ \text{e} & : & \text{dog(x)}\end{bmatrix}\\ \text{e} & : & \text{bark}(\mathfrak{c}.\text{x})\end{bmatrix}$$

This means we can restate (96) as (100).

(100) s : exist(dog′,bark′$|_{\mathcal{F}(dog')}$) iff

$$s : \left[\begin{array}{lll} \mathrm{x} & : & \mathfrak{T}(\mathrm{dog}') \\ \mathrm{e} & : & \left[\begin{array}{lll} \mathfrak{c} & : & \left[\begin{array}{lll} \mathrm{x}{=}\Uparrow^2\mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & \mathrm{dog}(\mathrm{x}) \end{array}\right] \\ \mathrm{e} & : & \mathrm{bark}(\mathfrak{c}.\mathrm{x}) \end{array}\right] \end{array}\right]$$

Note that the record type on the right hand side of (100) is truth-conditionally equivalent to the simpler record types in (101), that is, for any pair of the types there is a witness for one of the types just in case there is a witness for the other type.

(101) a. $\left[\begin{array}{lll} \mathrm{x} & : & \mathfrak{T}(\mathrm{dog}') \\ \mathrm{e} & : & \mathrm{bark}(\mathrm{x}) \end{array}\right]$

b. $\left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c} & : & \mathrm{dog}(\mathrm{x}) \\ \mathrm{e} & : & \mathrm{bark}(\mathrm{x}) \end{array}\right]$

The additional structure in (100) is unnecessary for this example but it will help us when we come to donkey anaphora.

The witness condition in (100) means that anything of the type resulting from applying (95) to a context will also be of the type (102).

(102) $$\left[\begin{array}{lll} \mathrm{restr}{=}\mathrm{dog}' & : & \mathit{Ppty} \\ \mathrm{scope}{=}\mathrm{bark}'\,|_{\mathfrak{F}(\mathrm{dog}')} & : & \mathit{Ppty} \\ \mathrm{e} & : & \left[\begin{array}{lll} \mathrm{x} & : & \mathfrak{T}(\mathrm{dog}') \\ \mathrm{e} & : & \left[\begin{array}{lll} \mathfrak{c} & : & \left[\begin{array}{lll} \mathrm{x}{=}\Uparrow^2\mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & \mathrm{dog}(\mathrm{x}) \end{array}\right] \\ \mathrm{e} & : & \mathrm{bark}(\mathfrak{c}.\mathrm{x}) \end{array}\right] \end{array}\right] \end{array}\right]$$

Assuming a theory of anaphora where anaphoric references must relate to paths in a witness, s, for the content, we could potentially have all of the examples of anaphora in (103).

(103) a. A dog barked. They (dogs in general, s.restr, MAXSET) do when they notice an intruder.

b. A dog barked. They (dogs in general which bark, s.scope) make such a racket.

c. A dog barked. That (an event of a dog barking, s.e, event anaphora) frightened off an intruder.

d. A dog barked. It (the dog which barked, s.e.x, individual, REFSET, anaphora) heard an intruder.

No dog barks The content for *no dog barks* would be derived in an exactly similar way as that for *a dog barks*. We obtain (104).

(104)
$$\ulcorner\lambda c{:}\left[\begin{array}{l} \textit{Cntxt} \\ \mathfrak{c}{:}\left[\begin{array}{l} \text{f:}\textit{PropCntxt} \\ \text{a:}\left[\begin{array}{l} \text{f:}\textit{PropCntxt} \\ \text{a:}\textit{PropCntxt} \end{array}\right] \end{array}\right] \end{array}\right].\left[\begin{array}{lcl} \text{restr=dog}' & : & \textit{Ppty} \\ \text{scope=bark}'\,|_{\mathfrak{F}(\text{dog}')} & : & \textit{Ppty} \\ \text{e} & : & \text{no(restr, scope)} \end{array}\right]\urcorner$$

Following the particular witness condition in (70), we can infer (105).

(105) $s : \text{no}(\text{dog}',\text{bark}'|_{\mathcal{F}(\text{dog}')})$ iff
$$s : \left[\begin{array}{lcl} \text{X} & : & \text{every}^{w}(\text{dog}') \\ \text{f} & : & ((x : \mathfrak{T}(X)) \rightarrow \neg\mathfrak{P}(\text{bark}'|_{\mathfrak{F}(\text{dog}')})\{x\}) \end{array}\right]$$

Given that $\mathfrak{P}(\text{bark}'|_{\mathfrak{F}(\text{dog}')})$ is (99) we can restate (105) as (106).

(106) $s : \text{no}(\text{dog}',\text{bark}'|_{\mathcal{F}(\text{dog}')})$ iff
$$s : \left[\begin{array}{lcl} \text{X} & : & \text{every}^{w}(\text{dog}') \\ \text{f} & : & ((x : \mathfrak{T}(X)) \rightarrow \neg\left[\begin{array}{lcl} \mathfrak{c} & : & \left[\begin{array}{lcl} \text{x=}\Uparrow^{2}\text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \end{array}\right] \\ \text{e} & : & \text{bark}(\mathfrak{c}.\text{x}) \end{array}\right]) \end{array}\right]$$

Anything of the type resulting from applying (104) to a context will also be of the type (107).

(107)
$$\left[\begin{array}{lcl} \text{restr=dog}' & : & \textit{Ppty} \\ \text{scope=bark}'\,|_{\mathfrak{F}(\text{dog}')} & : & \textit{Ppty} \\ \text{e} & : & \left[\begin{array}{lcl} \text{X} & : & \text{every}^{w}(\text{dog}') \\ \text{f} & : & ((x : \mathfrak{T}(X)) \rightarrow \neg\left[\begin{array}{lcl} \mathfrak{c} & : & \left[\begin{array}{lcl} \text{x=}\Uparrow^{2}\text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \end{array}\right] \\ \text{e} & : & \text{bark}(\mathfrak{c}.\text{x}) \end{array}\right]) \end{array}\right] \end{array}\right]$$

This suggests that we could potentially have the anaphora in (108).

(108) a. No dog barked. They (dogs in general, s.restr, MAXSET) normally do when they notice an intruder.

b. No dog barked. They (dogs in general which bark, s.scope) normally make such a racket.

c. No dog barked. That (an event of no dog barking, s.e, event anaphora) made it easy for the intruder.

d. No dog barked. They (the dogs which didn't bark, s.e.X, COMPSET anaphora) did not hear the intruder.

few dogs bark The content for an utterance of *few dogs bark* is either (109a) or (109b).

(109) a.
$$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{f:}\mathit{PropCntxt}\\ \text{a:}\begin{bmatrix}\text{f:}\mathit{PropCntxt}\\ \text{a:}\mathit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix}\,.\begin{bmatrix}\text{restr=dog}' & : & \mathit{Ppty}\\ \text{scope=bark}'\,|_{\mathfrak{F}(\text{dog}')} & : & \mathit{Ppty}\\ \text{e} & : & \text{few}_a(\text{restr, scope})\end{bmatrix}\urcorner$$

b.
$$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{f:}\mathit{PropCntxt}\\ \text{a:}\begin{bmatrix}\text{f:}\mathit{PropCntxt}\\ \text{a:}\mathit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix}\,.\begin{bmatrix}\text{restr=dog}' & : & \mathit{Ppty}\\ \text{scope=bark}'\,|_{\mathfrak{F}(\text{dog}')} & : & \mathit{Ppty}\\ \text{e} & : & \text{few}_p(\text{restr, scope})\end{bmatrix}\urcorner$$

Following the particular witness conditions in (85) and 86, we obtain (110a) and (110b) respectively.

(110) a. $s : \text{few}_a(\text{dog}', \text{bark}'|_{\mathcal{F}(\text{dog}')})$ iff
$$s : \begin{bmatrix}\text{X} & : & \overline{\text{few}_a^w(\text{dog}')}\\ \text{f} & : & ((x : \mathfrak{T}(X)) \to \neg\mathfrak{P}(\text{bark}'|_{\mathcal{F}(\text{dog}')})\{x\})\end{bmatrix}$$

b. $s : \text{few}_p(\text{dog}', \text{bark}'|_{\mathcal{F}(\text{dog}')})$ iff
$$s : \begin{bmatrix}\text{X} & : & \overline{\text{few}_p^w(\text{dog}')}\\ \text{f} & : & ((x : \mathfrak{T}(X)) \to \neg\mathfrak{P}(\text{bark}'|_{\mathcal{F}(\text{dog}')})\{x\})\end{bmatrix}$$

As with our treatment of *no* these can be unpacked as (111).

(111) a. $s : \text{few}_a(\text{dog}', \text{bark}'|_{\mathcal{F}(\text{dog}')})$ iff

$$s : \left[\begin{array}{lll} \text{X} & : & \overline{\text{few}_a^w(\text{dog}')} \\ \text{f} & : & ((x : \mathfrak{T}(X)) \to \neg \left[\begin{array}{lll} \mathfrak{c} & : & \left[\begin{array}{lll} \text{x}{=}\Uparrow^2\text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \end{array}\right] \\ \text{e} & : & \text{bark}(\mathfrak{c}\text{.x}) \end{array}\right]) \end{array}\right]$$

b. $s : \text{few}_p(\text{dog}', \text{bark}'|_{\mathcal{F}(\text{dog}')})$ iff

$$s : \left[\begin{array}{lll} \text{X} & : & \overline{\text{few}_p^w(\text{dog}')} \\ \text{f} & : & ((x : \mathfrak{T}(X)) \to \neg \left[\begin{array}{lll} \mathfrak{c} & : & \left[\begin{array}{lll} \text{x}{=}\Uparrow^2\text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \end{array}\right] \\ \text{e} & : & \text{bark}(\mathfrak{c}\text{.x}) \end{array}\right]) \end{array}\right]$$

Anything of the type resulting from applying the functions in (109) to a context will also be of the types in (112).

(112) a.

$$\left[\begin{array}{lll} \text{restr=dog}' & : & \textit{Ppty} \\ \text{scope=bark}'\,|_{\mathfrak{F}(\text{dog}')} & : & \textit{Ppty} \\ \text{e} & : & \left[\begin{array}{lll} \text{X} & : & \overline{\text{few}_a^w(\text{dog}')} \\ \text{f} & : & ((x : \mathfrak{T}(X)) \to \neg \left[\begin{array}{lll} \mathfrak{c} & : & \left[\begin{array}{lll} \text{x}{=}\Uparrow^2\text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \end{array}\right] \\ \text{e} & : & \text{bark}(\mathfrak{c}\text{.x}) \end{array}\right]) \end{array}\right] \end{array}\right]$$

b.

$$\left[\begin{array}{lll} \text{restr=dog}' & : & \textit{Ppty} \\ \text{scope=bark}'\,|_{\mathfrak{F}(\text{dog}')} & : & \textit{Ppty} \\ \text{e} & : & \left[\begin{array}{lll} \text{X} & : & \overline{\text{few}_p^w(\text{dog}')} \\ \text{f} & : & ((x : \mathfrak{T}(X)) \to \neg \left[\begin{array}{lll} \mathfrak{c} & : & \left[\begin{array}{lll} \text{x}{=}\Uparrow^2\text{x} & : & \textit{Ind} \\ \text{e} & : & \text{dog(x)} \end{array}\right] \\ \text{e} & : & \text{bark}(\mathfrak{c}\text{.x}) \end{array}\right]) \end{array}\right] \end{array}\right]$$

This suggests that we could potentially have the anaphora in (113).

(113) a. Few dogs barked. They (dogs in general, s.restr, MAXSET) normally do when they notice an intruder.

b. Few dogs barked. They (dogs in general which bark, s.scope) normally make such a racket.

c. Few dogs barked. That (an event of few dogs barking, s.e, event anaphora) made it easier for the intruder.

d. Few dogs barked. They (the dogs which didn't bark, s.e.X, COMPSET anaphora) did not hear the intruder.

7.5 Long distance dependencies

Let us consider how to derive the content of *who Sam hugged* as a relative clause. We use $\mathbf{hug}_V$ to represent the content of *hug* as a transitive verb, characterized as in (114).

(114) $\mathbf{hug}_V = \ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda\mathcal{Q}{:}\mathit{Quant}\ .\ \ulcorner\lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right]\ .\ \left[\mathrm{e}{:}\mathrm{hug}(r.\mathrm{x},\ \mathcal{Q})\right]\urcorner\urcorner$

Our theory of syntax in terms of event types means that what have been called "gaps" or "traces" in other theories of syntax would somehow have to correspond to non-events in which nothing happens and do not have any temporal extent. This strongly suggests that they do not exist. Instead we will take a strategy similar to that pursued, for example, in combinatory categorial grammar (see, for example, Steedman, 2012) where *hug* can also be interpreted as a verb phrase whose content is "looking for" a *wh*-phrase content. We will add a third component to our contexts with label '$\mathfrak{g}$' for assignments associated with gaps. Thus we will redefine the type *Cntxt* as (115).

(115) $\begin{bmatrix} \mathfrak{s} & : & \mathit{Assgnmnt} \\ \mathfrak{g} & : & \mathit{Assgnmnt} \\ \mathfrak{c} & : & \mathit{PropCntxt} \end{bmatrix}$

We will represent this content as $\mathbf{hug}_{VP}$ and characterize it in terms of $\mathbf{hug}_V$ as in (116a) which is identical with (116b).

(116) a. $\mathbf{hug}_{VP} = \ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{g}{:}\left[\mathrm{x}_0{:}\mathit{Ind}\right]\end{bmatrix}\ .\ \mathbf{hug}_V(c)(\lambda P{:}\mathit{Ppty}\ .\ P(\left[\mathrm{x}{=}c.\mathfrak{g}.\mathrm{x}_0\right]))\urcorner$

b. $\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{g}{:}\left[\mathrm{x}_0{:}\mathit{Ind}\right]\end{bmatrix}\ .\ \ulcorner\lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right]\ .\ \left[\mathrm{e}{:}\mathrm{hug}(r.\mathrm{x},\ \lambda P{:}\mathit{Ppty}\ .\ P(\left[\mathrm{x}{=}c.\mathfrak{g}.\mathrm{x}_0\right]))\right]\urcorner\urcorner$

Let us now consider the content of *Sam hugged*. The parametric content of *Sam*, which we will represent as **Sam**, is (117) .

(117) $\mathbf{Sam} = \ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\\ \mathrm{e}{:}\mathrm{named}(\mathrm{x},\ \text{“Sam”})\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\mathit{Ppty}\ .\ P(c.\mathfrak{c})\urcorner$

In order to combine **Sam** with $\mathbf{hug}_{VP}$ we will need to adjust the combination of parametric contents based on functional application so that it takes account of incrementation of the context path $\mathfrak{w}.\mathrm{x}$ in addition to $\mathfrak{s}.\mathrm{x}$ to allow for more than one *wh*-dependency. This uses the operation 'incr' defined in Chapter 4, p. 204. We will use (118a) to represent (118b).

(118) a. $\mathrm{incr}_{\pi_1,\ldots,\pi_n}(\varphi, T)$

b. $\mathrm{incr}_{\pi_1}(\ldots\mathrm{incr}_{\pi_n}(\varphi, T), \ldots, T)$

We will informally use (119) to represent incrementation of all paths which lead to assignments so that we do not have to keep track of the required subscripts as we add additional assignments to our contexts.

(119) $\mathrm{incr}(\varphi, T)$

We now modify the definition of the combination of α and β, $\alpha @ \beta$, given in Chapter 4, p. 204 to (120).

(120) If $\alpha : \begin{bmatrix} \mathrm{bg}{:}\mathit{CntxtType} \\ \mathrm{fg}{:}(\mathrm{bg}{\rightarrow}(T_1 \rightarrow T_2)) \end{bmatrix}$ and $\beta : \begin{bmatrix} \mathrm{bg}{:}\mathit{CntxtType} \\ \mathrm{fg}{:}(\mathrm{bg}{\rightarrow}\ T_1) \end{bmatrix}$ then the *combination of* α *and* β *based on functional application*, $\alpha @ \beta$, is

$$\ulcorner\lambda c{:}[\alpha.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}} \underset{.}{\wedge} \mathrm{incr}([\beta.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}, \alpha.\mathrm{bg})\ .\ [\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}}(c)(\mathrm{incr}([\beta.\mathrm{fg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}, \alpha.\mathrm{bg})(c))\urcorner$$

The content of *Sam hugged*, which we will represent as **Sam⌢hugged**, is **Sam@hug**$_{VP}$, that is (121a) which is identical with (121b).

(121) a. **Sam⌢hugged =**

$$\ulcorner\lambda c{:}\begin{bmatrix} \mathit{Cntxt} \\ \mathfrak{g}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix} \\ \mathfrak{c}{:}\begin{bmatrix} \mathrm{f}{:}\begin{bmatrix} \mathit{PropCntxt} \\ \mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Sam”}) \end{bmatrix} \\ \mathrm{a}{:}\mathit{PropCntxt} \end{bmatrix} \end{bmatrix}\ .\ \mathbf{Sam}_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}}(c)(\mathbf{hug}_{VP,\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}(c))\urcorner$$

b. $$\ulcorner\lambda c{:}\begin{bmatrix} \mathit{Cntxt} \\ \mathfrak{g}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix} \\ \mathfrak{c}{:}\begin{bmatrix} \mathrm{f}{:}\begin{bmatrix} \mathit{PropCntxt} \\ \mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Sam”}) \end{bmatrix} \\ \mathrm{a}{:}\mathit{PropCntxt} \end{bmatrix} \end{bmatrix}\ .\ \begin{bmatrix}\mathrm{e}{:}\mathrm{hug}(c.\mathrm{f.x}, \lambda P{:}\mathit{Ppty}\ .\ P(\begin{bmatrix}\mathrm{x}{=}c.\mathfrak{g}.\mathrm{x}_0\end{bmatrix}))\end{bmatrix}\urcorner$$

In order to treat *wh*-phrases we will add a fourth component to contexts labelled with ‘$\mathfrak{w}$’ for an assignment. Thus we revise *Cntxt* to (122).

(122) $\begin{bmatrix} \mathfrak{s} & : & \mathit{Assgnmnt} \\ \mathfrak{w} & : & \mathit{Assgnmnt} \\ \mathfrak{g} & : & \mathit{Assgnmnt} \\ \mathfrak{c} & : & \mathit{PropCntxt} \end{bmatrix}$

The content of *who*, which we will represent as **who**, is given in (123).

(123) $\textbf{who} = \ulcorner \lambda c{:}\begin{bmatrix}\mathit{Cntxt} \\ \mathfrak{w}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix}\end{bmatrix} .\ \lambda P{:}\mathit{Ppty}\ .\ P(\begin{bmatrix}\mathrm{x}{=}c.\mathfrak{w}.\mathrm{x}_0\end{bmatrix})\urcorner$

This content is identical with the universal resource ‘SemWhPron’ and is exactly parallel to ‘SemPron’ but using the context label ‘$\mathfrak{w}$’ rather than ‘$\mathfrak{s}$’. There will also be a universal lexical resource ‘LexWhPron’ parallel to ‘LexPron’, characterized in (124).

(124) If T_{phon} is a phonological type, then LexWhPron(T_{phon}) is

$$\mathrm{Lex}(T_{\mathrm{phon}}, \mathit{NP}) \mathbin{\dot{\wedge}} \begin{bmatrix}\mathrm{cont{=}SemWhPron}{:}\mathit{PQuant}\end{bmatrix}$$

(124) makes a *wh*-word such as *who* into a noun phrase and this is important, for instance, if it occurs in a noun phrase position such as the subject or object of a sentence. However, these phrases also have a role as complementizers as *who* does in relative clauses, that is in a fronted position in the sentence and not in a argument position to a verb or preposition. It is important in this case to recognize that this is a *wh*-NP, that is, an NP with a *wh*-content. For this we have the type in (125) which we can abbreviate as *WhNP*.

(125) $\begin{bmatrix} \mathit{NP} & & \\ \mathrm{cont{=}SemWhPron} & : & \mathit{PQuant} \end{bmatrix}$

(125) would, however, only allow the wh-index introduced by ‘SemWhPron’, that is ‘0´ and not allowing for incremented versions of this obtained by combining constituents. A more general way of characterizing the type *WhNP* is to treat it as a basic type with the witness condition in (126).

(126) σ : *WhNP* iff σ : *NP*, σ.cont is $\mathcal{Q}$ and $\mathcal{Q}$.bg $\sqsubseteq \begin{bmatrix}\mathfrak{w}{:}\begin{bmatrix}\mathrm{x}_i{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$, for some natural number i.

In order to be able to characterize the kind of binding involved in combining *who* with *Sam hugged* we will use the notation $r[\pi = v]$ for a record, r, path, π, and value v. This notation is characterized in (127).

(127) If r is a record, π is $\ell_0.\ell_1.\ldots.\ell_n$ where $\ell_0, \ell_1, \ldots, \ell_n$ are labels and v is an object of some type, then

$r[\pi = v]$ is the record, r', exactly like r except for the possible difference that $r'.\pi = v$.

We can make (127) more explicit as in (128), repeated in Appendix 11.1.

(128) If r is a record, π is $\ell_0.\ell_1.\ldots.\ell_n$ where $\ell_0, \ell_1, \ldots, \ell_n$ are labels and v is an object of some type, then

if $\pi \notin \mathrm{paths}(r)$, then $r[\pi = v]$ is the smallest record, r', such that

1. $\pi \in \mathrm{paths}(r')$
2. $r'.\pi = v$
3. for any $\pi' \in \mathrm{paths}(r)$, $\pi' \in \mathrm{paths}(r')$ and $r'.\pi' = r.\pi'$

if $\pi \in \mathrm{paths}(r)$, then $r[\pi = v]$ is the smallest record, r', such that

1. $\pi \in \mathrm{paths}(r')$ and $r'.\pi = v$
2. for any $\pi' \in (\mathrm{paths}(r) - \{\pi'' \mid \pi'' \leq \pi\})$, $\pi' \in \mathrm{paths}(r')$ and $r'.\pi' = r.\pi'$

Recall that $\pi_1 \leq \pi_2$ as used in (128) means that π_1 is an initial subpath of π_2 (see Chapter 4, p. 178).

We generalize this notation so that we can use a record to modify another record. This is characterized in (129), where $\mathrm{tpaths}(r)$ represents the set of total paths in r (Chapter 1,p. 33).

(129) If r_1 is a record and r_2 is a record such $\mathrm{tpaths}(r_2) = \{\pi_1, \ldots, \pi_n\}$, then $r_1[r_2]$ is $r_1[\pi_1 = r_2.\pi_1] \ldots [\pi_n = r_2.\pi_n]$

This now gives us what we need to characterize the content of the relative clause *who Sam hugged*, which we will represent as **who⌢Sam⌢hugged**, as in (130a) which is identical with (130b). (130b) is in turn equivalent to (130c) because of the constraint on the extensional predicate ‘hug’ similar to that for ‘find’ given in (65) on p. 316.

(130) a. **who⌢Sam⌢hugged =**

$$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{c}{:}\left[\begin{array}{l}\text{f:}\textit{PropCntxt}\\ \text{a:}\left[\begin{array}{l}\text{f:}\left[\begin{array}{l}\textit{PropCntxt}\\ \text{x:}\textit{Ind}\\ \text{e:named(x, ``Sam'')}\end{array}\right]\\ \text{a:}\textit{PropCntxt}\end{array}\right]\end{array}\right]\end{array}\right].$$

$$\ulcorner\lambda r_1{:}\left[\text{x:}\textit{Ind}\right].$$

$$\textbf{who}_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}}(c[\mathfrak{w}.\mathrm{x}_0 = r_1.\mathrm{x}])\ (\ulcorner\lambda r_2{:}\left[\text{x:}\textit{Ind}\right].\ \textbf{Sam}{}^\frown\textbf{hugged}_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}(c[\mathfrak{g}.\mathrm{x}_0 = r_2.\mathrm{x}])\urcorner)\urcorner\urcorner$$

b. $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{c}{:}\left[\begin{array}{l}\text{f:}\textit{PropCntxt}\\ \text{a:}\left[\begin{array}{l}\text{f:}\left[\begin{array}{l}\textit{PropCntxt}\\ \text{x:}\textit{Ind}\\ \text{e:named(x, ``Sam'')}\end{array}\right]\\ \text{a:}\textit{PropCntxt}\end{array}\right]\end{array}\right]\end{array}\right].$$

$$\ulcorner\lambda r_1{:}\left[\text{x:}\textit{Ind}\right].\ \left[\text{e:hug}(c.\mathfrak{c}.\text{a.f.x},\ \lambda P{:}\textit{Ppty}\ .\ P(\left[\text{x}{=}r_1.\text{x}\right]))\right]\urcorner\urcorner$$

c. $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{c}{:}\left[\begin{array}{l}\text{f:}\textit{PropCntxt}\\ \text{a:}\left[\begin{array}{l}\text{f:}\left[\begin{array}{l}\textit{PropCntxt}\\ \text{x:}\textit{Ind}\\ \text{e:named(x, ``Sam'')}\end{array}\right]\\ \text{a:}\textit{PropCntxt}\end{array}\right]\end{array}\right]\end{array}\right].\ \ulcorner\lambda r_1{:}\left[\text{x:}\textit{Ind}\right].\ \left[\text{e:hug}^\dagger(c.\mathfrak{c}.\text{a.f.x},\ r_1.\text{x}))\right]\urcorner\urcorner$$

In order to achieve this we need a general operation of *wh*-combination which will pass whatever context restrictions are associated with the *wh*-phrase and those that are associated with the phrase with which it is being combined up to context restrictions associated with the whole phrase except for those associated with the paths '$\mathfrak{w}.\mathrm{x}_i$' (for the *wh*-phrase which is binding the gap) and '$\mathfrak{g}.\mathrm{x}_j$' (for the gap being bound). The discharging of the requirements associated with these two paths will involve subtracting them from the types which are passed up to the context type for the combined phrase.

As record types are labelled sets (Chapter 1, p. 34ff.), this subtraction is based on set-theoretic subtraction.

We first define what it means to subtract the field labelled ℓ from an assignment type, T, in (131).

(131) If $T \sqsubseteq$ *Assgnmnt* and $\ell \in \mathrm{labels}(T)$, then

1. if $\mathrm{labels}(T) = \{\ell\}$, then $T \ominus \ell =$ *Assgnmnt*
2. otherwise, $T \ominus \ell = T - \{\langle \ell, \varphi\rangle\}$, where $\langle \ell, \varphi\rangle \in T$.

Based on (131) we can define the subtraction of a path to an assignment in a context type given in (132).

(132) (preliminary)

$$\text{If } T \sqsubseteq \begin{bmatrix} \textit{Cntxt} \\ \ell_1 : \begin{bmatrix} \textit{Assgnmnt} \\ \ell_2 : \textit{Ind} \end{bmatrix} \end{bmatrix}, \text{ then } T \ominus \ell_1.\ell_2 \text{ is } T \boxed{\wedge} [\ell_1 : (T.\ell_1 \ominus \ell_2)]$$

This is a preliminary definition because it does not take account of what happens if the path that is being removed has dependent fields associated with it. Suppose, for example, that the context type associated with *who* were (133) where we represent the presupposition that what is associated with *who* is a person.

(133)
$$\begin{bmatrix} \textit{Cntxt} & & \\ \mathfrak{w} & : & \begin{bmatrix} \mathrm{x}_0 & : & \textit{Ind} \end{bmatrix} \\ \mathfrak{c} & : & \begin{bmatrix} \mathrm{e} & : & \mathrm{person}(\Uparrow\mathfrak{w}.\mathrm{x}_0) \end{bmatrix} \end{bmatrix}$$

Suppose that T is (133). Then $T \ominus \mathfrak{w}.\mathrm{x}_0$ according to our present definition would be (134) which is not a record type since there is a dependence on '$\mathfrak{w}.\mathrm{x}_0$' but there is no such path.

(134)
$$\begin{bmatrix} \textit{Cntxt} & & \\ \mathfrak{w} & : & \textit{Assgnmnt} \\ \mathfrak{c} & : & \begin{bmatrix} \mathrm{e} & : & \mathrm{person}(\Uparrow\mathfrak{w}.\mathrm{x}_0) \end{bmatrix} \end{bmatrix}$$

The solution to this is to extend subtraction so that it removes not only the path referred to but also those paths that depend on it. Thus the result of $T \ominus \mathfrak{w}.\mathrm{x}_0$ should rather be (135).

(135)
$$\begin{bmatrix} \textit{Cntxt} & & \\ \mathfrak{w} & : & \textit{Assgnmnt} \\ \mathfrak{c} & : & \textit{PropCntxt} \end{bmatrix}$$

In this particular case, the resulting type, (135), is actually identical with *Cntxt*.

Actually, things are more complicated that this. For any path we remove by this method we must recursively remove all the dependencies on that path. Thus if something had depended on '$\mathfrak{c}$.e' in the above example we would have had to remove that and so on. In our context types, as defined here, all the dependencies on assignments will occur under the label '$\mathfrak{c}$', that is, in the propositional context. It is perhaps most straightforward to use the unique identifier notation introduced in Chapter 2, p. 79, in order to identify paths which depend on another path. Consider again (134). In unique identifier notation this is (136).

(136) $$\left[\begin{array}{lll} \textit{Cntxt} & & \\ \mathfrak{w} & : & \left[\begin{array}{lll} \mathrm{x}_{0_{\boxed{0}}} & : & \textit{Ind} \end{array}\right] \\ \mathfrak{c} & : & \left[\begin{array}{lll} \mathrm{e} & : & \mathrm{person}(\boxed{0}) \end{array}\right] \end{array}\right]$$

In (136), the path '$\mathfrak{c}$.e' depends on '$\mathfrak{w}.\mathrm{x}_0$' because the representation of the type at the former contains the unique identifier which indexes the latter. We give a general characterization of dependency between paths in a record type in (137) (repeated in Appendix 14).

(137) If T is a record type, $\pi_1 \in \mathrm{paths}(T)$ and $\pi_2 \in \mathrm{tpaths}(T)$, then π_2 *depends on* π_1 iff, in unique identifier notation, π_1 is indexed with $\boxed{i}$ and the representation of $T.\pi_2$ contains $\boxed{i}$, for some natural number i.

Given this notion of dependency it is then possible to define the set of paths in a record type, T, which constitute a dependency family on some particular path, π, which we will represent as $\mathrm{paths}_\pi(T)$. This is given in (138) (repeated in Appendix 14).

(138) If T is a record type and $\pi \in \mathrm{paths}(T)$ then *the dependency family of* π *in* T, $\mathrm{paths}_\pi(T)$, is that subset, Π, of $\mathrm{paths}(T)$ such that

1. $\pi \in \Pi$
2. for any $\pi' \in \Pi$ and $\pi'' \in \mathrm{tpaths}(T)$, if π'' depends on π', then $\pi'' \in \Pi$
3. for and $\pi' \in \Pi$ and $\pi'' \in \mathrm{paths}(T)$, if π' depends on π'', then $\pi'' \in \Pi$

We use π/T as an shorter alternative notation for $\mathrm{paths}_\pi(T)$.

That is, the dependency family of π includes π itself, anything that depends on anything in the dependency family and anything on which something in the dependency family depends. This dependency family gives rise to a generalization of the original type, T, which we will call T *generalized to* π, T^π. We define this in (139) (repeated in Appendix 14).

(139) If T is a record type and $\pi \in \mathrm{paths}(T)$, then T *generalized to* π, T^π, is the smallest labelled set T' such that $\mathrm{paths}_\pi(T) \subseteq \mathrm{paths}(T')$ and for all $\pi' \in \mathrm{tpaths}(T')$, $T'.\pi = T.\pi$

We give a definition of a more general version of (139) in (140) (repeated in Appendix 14) where we generalize to several paths in a type simultaneously.

(140) If T is a record type and $\{\pi_1, \ldots, \pi_n\} \subset \text{paths}(T)$, then T *generalized to* $\pi_1, \ldots, \pi_n$, $T^{\pi_1,\ldots,\pi_n}$, is the smallest labelled set T' such that for all $\pi_i \in \{\pi_1, \ldots, \pi_n\}$, $\text{paths}_{\pi_i}(T) \subseteq \text{paths}(T')$ and for all $\pi' \in \text{tpaths}(T')$, $T'.\pi = T.\pi$

This notion of generalization is important if we wish to associate presuppositions with *wh*-phrases and extract them from the context type for local accommodation at the point at which the *wh*-phrase binds a gap.

We can now generalize the subtraction operation, $\ominus$. First we will characterize branching paths in a labelled set, as in (141).

(141) If X is a labelled set and $\pi \in \text{tpaths}(X)$ then π *is branching in* X iff

1. there is some $\pi' \in \text{tpaths}(X)$ such that $\pi' \neq \pi$
2. there is some π'' such that $\pi'' < \pi'$ and $\pi'' < \pi$

(Recall that $\pi_1 < \pi_2$ means that π_1 is a proper initial subpath of π_2, Chapter 4, p. 178.)

We characterize subtraction in general for labelled sets as (142) (repeated in Appendix 1).

(142) a. If X is a labelled set, $\ell \in \text{labels}(X)$ and $\langle \ell, \varphi \rangle \in X$, then

$$X \ominus \ell = X - \{\langle \ell, \varphi \rangle\}$$

b. If X is a labelled set, $\langle \ell, \varphi \rangle \in X$ and $\ell.\pi \in \text{tpaths}(X)$, then

if $\ell.\pi$ is branching in X, then

$$X \ominus \ell.\pi = (X - \{\langle \ell, \varphi \rangle\}) \cup \{\langle \ell, \varphi \ominus \pi \rangle\}$$

otherwise

$$X \ominus \ell.\pi = X - \{\langle \ell, \varphi \rangle\}$$

If X is a labelled set and $\pi_1, \ldots, \pi_n \in \text{tpaths}(X)$, then we write (143a) for (143b)

(143) a. $X \ominus \pi_1, \ldots, \pi_n$

b. $X \ominus \pi_1 \ominus \ldots \ominus \pi_n$

If T is a record type, $\pi \in \text{paths}(T)$ and $\text{paths}_\pi(T) = \{\pi, \pi_1, \ldots, \pi_n\}$, then we normally write $T \ominus \pi/T$ for $T \ominus \pi, \pi_1, \ldots, \pi_n$.

We now extend subtraction to include some cases where it would otherwise be undefined.

(144)
1. If T is *Assgnmnt*$\wedge T'$, T' is a record type and $\ell \in \text{labels}(T')$, then
   a) if $\text{labels}(T') = \{\ell\}$, $T \ominus \ell =$ *Assgnmnt*
   b) otherwise, $T \ominus \ell =$ *Assgnmnt* $\wedge\ (T' \ominus \ell/T)$
2. If T is *PropCntxt*$\wedge T'$ and $\pi \in \text{tpaths}(T')$ then
   a) if π is ℓ and $\text{labels}(T') = \{\ell\}$, then $T \ominus \pi =$ *PropCntxt*
   b) otherwise, $T \ominus \pi =$ *PropCntxt* $\wedge\ (T' \ominus \pi/T')$

We can now create a preliminary definition of combination based on functional application for *wh*-phrases and sentences with gaps, '$@_{\text{wh}}$', characterized in (145).

(145) (preliminary)

If

1. $\alpha : \begin{bmatrix}\text{bg:}\textit{CntxtType}\\ \text{fg:(bg}\rightarrow\textit{Quant}\text{)}\end{bmatrix}$,
2. $\beta : \begin{bmatrix}\text{bg:}\textit{CntxtType}\\ \text{fg:(bg}\rightarrow\textit{RecType}\text{)}\end{bmatrix}$,
3. $\alpha.\text{bg} \sqsubseteq \begin{bmatrix}\mathfrak{w}\text{:}\begin{bmatrix}\text{x}_i\text{:}\textit{Ind}\end{bmatrix}\end{bmatrix}$ for some natural number, i, and
4. $\beta.\text{bg} \sqsubseteq \begin{bmatrix}\mathfrak{g}\text{:}\begin{bmatrix}\text{x}_j\text{:}\textit{Ind}\end{bmatrix}\end{bmatrix}$ for some natural number, j,

then *the wh$_{i,j}$-combination of* α *and* β, $\alpha @_{\text{wh}_{i,j}} \beta$, is

$$\ulcorner\lambda c\text{:}([\alpha.\text{bg}\ominus\mathfrak{w}.\text{x}_i]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}\dot{\wedge}\text{incr}([\beta.\text{bg}\ominus\mathfrak{g}.\text{x}_j]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.a}, \alpha.\text{bg}))\ .$$
$$\mathfrak{P}(\ulcorner\lambda r_1\text{:}\begin{bmatrix}\text{x:}\textit{Ind}\\ \mathfrak{w}\text{:}\begin{bmatrix}\text{x}_i\text{=}\Uparrow\text{x:}\textit{Ind}\end{bmatrix}\end{bmatrix}\ .$$
$$\alpha_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}(c[r_1])\ (\mathfrak{P}(\ulcorner\lambda r_2\text{:}\begin{bmatrix}\text{x:}\textit{Ind}\\ \mathfrak{g}\text{:}\begin{bmatrix}\text{x}_j\text{=}\Uparrow\text{x:}\textit{Ind}\end{bmatrix}\end{bmatrix}\ .$$
$$\text{incr}(\beta_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}, \alpha.\text{bg})(c[r_2])\urcorner))\urcorner)\urcorner$$

Recall that $\mathfrak{P}(P)$ represents the purification of the property P defined in (12) on p. 354.

We can adjust (145) to take account of any presuppositions introduced by the *wh* and gap interpretations as in (146).

(146) If

1. $\alpha : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow\mathit{Quant})\end{bmatrix}$,
2. $\beta : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow\mathit{RecType})\end{bmatrix}$,
3. $\alpha.\text{bg} \sqsubseteq \begin{bmatrix}\mathfrak{w}\text{:}\begin{bmatrix}\text{x}_i\text{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$ for some natural number, i, and
4. $\beta.\text{bg} \sqsubseteq \begin{bmatrix}\mathfrak{g}\text{:}\begin{bmatrix}\text{x}_j\text{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$ for some natural number, j,

then *the wh*$_{i,j}$*-combination of* α *and* β, $\alpha@_{\text{wh}_{i,j}}\beta$, is

$$\begin{array}{l}\ulcorner\lambda c\text{:}([\alpha.\text{bg}\ominus\text{paths}_{\mathfrak{w}.\text{x}_i}(\alpha.\text{bg})]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}\dot{\wedge}\\ \qquad\qquad\quad \text{incr}([\beta.\text{bg}\ominus\text{paths}_{\mathfrak{g}.\text{x}_j}(\beta.\text{bg})]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.a},\alpha.\text{bg}))\ .\\ \quad \mathfrak{P}(\ulcorner\lambda r_1\text{:}[\alpha.\text{bg}^{\mathfrak{w}.\text{x}_i}]_{\mathfrak{w}.\text{x}_i\rightsquigarrow\text{x}}\ .\\ \qquad\quad \alpha_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f},\mathfrak{w}.\text{x}_i\rightsquigarrow\text{x}}(c[r_1])\ (\mathfrak{P}(\ulcorner\lambda r_2\text{:}[\beta.\text{bg}^{\mathfrak{g}.\text{x}_j}]_{\mathfrak{g}.\text{x}_j\rightsquigarrow\text{x}}\ .\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{incr}(\beta_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a},\mathfrak{g}.\text{x}_j\rightsquigarrow\text{x}},\alpha.\text{bg})(c[r_2])\urcorner))\urcorner)\urcorner\end{array}$$

The effect of the property purification here will be to locally accommodate whatever presuppositions are associated ‘$\mathfrak{w}.\text{x}_i$’ or ‘$\mathfrak{g}.\text{x}_j$’ at the level at which the *wh*-binding takes place.

We leave the treatment of contraints on long distance dependencies like *wh*-constructions (such as *island constraints*) to future work. The strategy for handling these would be to elaborate on the conditions 1.–4. in (146) or as conditions associated with constituent structure rules.

We can then define a version of ‘ContForwardApp’ (forward application of contents) based on wh$_{i,j}$-combination as given in (147).

(147) If T_{fun}, T_{arg} and T_{res} are types such that if $\alpha : T_{\text{fun}}$ and $\beta : T_{\text{arg}}$, then $\alpha@_{\text{wh}_{i,j}}\beta$ is defined and of type T_{res}, then

$$\text{ContForwardApp}_{@_{\text{wh}_{i,j}}}(T_{\text{fun}}, T_{\text{arg}}, T_{\text{res}})$$

is

$$\lambda u\text{:}\begin{bmatrix}\text{cont:}T_{\text{fun}}\end{bmatrix}^\frown\begin{bmatrix}\text{cont:}T_{\text{arg}}\end{bmatrix}\ .\ \begin{bmatrix}\text{cont=}u[0].\text{cont}@_{\text{wh}_{i,j}}u[1].\text{cont:}T_{\text{res}}\end{bmatrix}$$

We can then characterize a notation for phrase structure rules which use ‘ContForwardApp$_{@_{\mathrm{wh}_{i,j}}}$’ as in (148).

(148) If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types, $T_{\mathrm{daughter}_1} \sqsubseteq \begin{bmatrix}\mathrm{cont}{=}c_1{:}\mathit{Cont}\end{bmatrix}$ where $c_1.\mathrm{bg} \sqsubseteq \begin{bmatrix}\mathfrak{w}{:}\begin{bmatrix}\mathrm{x}_i{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$, $T_{\mathrm{daughter}_2} \sqsubseteq \begin{bmatrix}\mathrm{cont}{=}c_2{:}\mathit{Cont}\end{bmatrix}$ where $c_2.\mathrm{bg} \sqsubseteq \begin{bmatrix}\mathfrak{g}{:}\begin{bmatrix}\mathrm{x}_j{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$ and T_{arg} and T_{res} are content types, then

$$T_{\mathrm{mother}} \longrightarrow T_{\mathrm{daughter}_1}\ T_{\mathrm{daughter}_2} \ \mid\ T'_{\mathrm{daughter}_1}(@_{\mathrm{wh}_{i,j}} T'_{\mathrm{daughter}_2} : T_{\mathrm{arg}}) : T_{\mathrm{res}}$$

is

$$T_{\mathrm{mother}} \longrightarrow T_{\mathrm{daughter}_1}\ T_{\mathrm{daughter}_2} \wedge\!\!\!\!\cdot\ \ \mathrm{ContForwardApp}_{@_{\mathrm{wh}_{i,j}}}(T_{\mathrm{arg}}, T_{\mathrm{res}})$$

We introduce a sign type *Rel* which requires the category to be ‘rel’ for relative clause and types $\mathit{NP}_{\mathrm{wh}_i}$ and S/i whose witness conditions are given in (149).

(149) a. $\alpha : \mathit{NP}_{\mathrm{wh}_i}$ iff $\alpha : \mathit{NP}$ and $\alpha.\mathrm{cont.bg} \sqsubseteq \begin{bmatrix}\mathfrak{w}{:}\begin{bmatrix}\mathrm{x}_i{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$

b. $\alpha : \mathit{S}/i$ iff $\alpha : \mathit{S}$ and $\alpha.\mathrm{cont.bg} \sqsubseteq \begin{bmatrix}\mathfrak{g}{:}\begin{bmatrix}\mathrm{x}_i{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$

The slash notation ‘S/i’ is derived from the use of slash categories to represent constituents containing gaps in Generalized Phrase Structure Grammar (Gazdar *et al.*, 1985) and Head Driven Phrase Structure Grammar (Sag *et al.*, 2003).

We can now add the rule in (150) to the language specific resources for English.

(150) $\mathit{Rel} \longrightarrow \mathit{NP}_{\mathrm{wh}_i}\ \mathit{S}/j \mid \mathit{NP}'_{\mathrm{wh}_i}(@_{\mathrm{wh}_{i,j}} \mathit{S}/j'{:}\mathit{PRecType}){:}\mathit{PPpty}$

To form the content of a noun modified by a relative clause such as *child who Sam hugged* we use a record type which requires of an individual that it has both the property of being a child and being hugged by Sam as given in (151).

(151)
$$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\mathit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\mathit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\begin{bmatrix}\mathit{PropCntxt}\\ \mathrm{x}{:}\mathit{Ind}\\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Sam”})\end{bmatrix}\\ \mathrm{a}{:}\mathit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}\ .$$
$$\ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\end{bmatrix}\ .$$

$$\left[\begin{array}{lll} \mathrm{e}_1 & : & \text{child}'\{r.\mathrm{x}\} \\ \mathrm{e}_2 & : & \textbf{who}\frown\textbf{Sam}\frown\textbf{hugged}_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{a}}(c)\{r.\mathrm{x}\} \end{array}\right]\urcorner\urcorner$$

In order to achieve this we will first introduce an operation of property conjunction in (152). As a preliminary, we introduce types for properties and parametric properties of particular types of objects. This is done in (152).

(152) a. If T is a type, then ^{T}*Ppty* is a type – "the type of properties of objects of type T"

b. P : ^{T}*Ppty* iff P : *Ppty* and P.bg $\sqsubseteq \left[\mathrm{x}{:}T\right]$

c. If T is a type, then ^{T}*PPpty* is a type – "the type of parametric properties of objects of type T"

d. $\mathcal{P}$: ^{T}*PPpty* iff $\mathcal{P}$: *PPpty* and for any c : $\mathcal{P}$.bg, $\mathcal{P}(c)$: ^{T}*Ppty*

We can now characterize property conjunction as in (153).

(153) If T is a type, P_1 : ^{T}*Ppty* and P_2 : ^{T}*Ppty*, then *the conjunction of* P_1 *and* P_2, $P_1\&P_2$, is

$$\ulcorner\lambda r{:}\left[\mathrm{x}{:}T\right] . \left[\begin{array}{lll} \mathrm{e}_1 & : & P_1\{r.\mathrm{x}\} \\ \mathrm{e}_2 & : & P_2\{r.\mathrm{x}\} \end{array}\right]\urcorner$$

We can now characterize a combination operation, $@_{\&}$, in (154).

(154) If T is a type, α : ^{T}*PPpty* and β : ^{T}*PPpty* then *the property conjunction combination of* α *and* β, $\alpha @_{\&} \beta$, is

$$\lambda c{:}[\alpha.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}} \dot{\wedge} \mathrm{incr}([\beta.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}, \alpha.\mathrm{bg}) \;.\; \alpha_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}}(c) \& \mathrm{incr}([\beta]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}, \alpha.\mathrm{bg})(c)$$

We introduce a version of 'ContForwardApp' for '$@_{\&}$' in (155).

(155) If T is a type, then ContForwardApp$_{@_{\&}}(T)$ is

$$\lambda u{:}\left[\ \text{cont} \ : \ {}^{T}\textit{PPpty}\ \right]\frown\left[\ \text{cont} \ : \ {}^{T}\textit{PPpty}\ \right] .$$
$$\left[\ \text{cont}{=}u[0].\text{cont}@_{\&}u[1].\text{cont} \ : \ {}^{T}\textit{PPpty}\ \right]$$

We can then introduce a notation of constituent structure rules which use 'ContForwardApp$_{@_{\&}}$' in (156).

(156) If T is a type, T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types, $T_{\text{daughter}_1} \sqsubseteq \left[\ \text{cont} \ : \ {}^{T}\mathit{PPpty}\ \right]$ and $T_{\text{daughter}_2} \sqsubseteq \left[\ \text{cont} \ : \ {}^{T}\mathit{PPpty}\ \right]$, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \ \mid\ T'_{\text{daughter}_1}(@_{\&} T'_{\text{daughter}_2} : {}^{T}\mathit{PPpty}) : {}^{T}\mathit{PPpty}$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \mathbin{\overset{\wedge}{.}} \text{ContForwardApp}_{@_{\&}}(T)$$

Finally, we introduce the English rules for combining common nouns with relative clauses. First we introduce types ${}^{T}\mathit{N}$ and ${}^{T}\mathit{Rel}$ in (157).

(157) a. If T is a type, then ${}^{T}\mathit{N}$ and ${}^{T}\mathit{Rel}$ are types.
b. α : ${}^{T}\mathit{N}$ or ${}^{T}\mathit{Rel}$ iff α : *N* or *Rel* and α.cont : ${}^{T}\mathit{PPpty}$

For any type, T, we introduce the constituent structure rule in (158).

(158) $\mathit{N} \longrightarrow {}^{T}\mathit{N}\ {}^{T}\mathit{Rel} \ \mid\ {}^{T}\mathit{N}'(@_{\&}{}^{T}\mathit{Rel}' : {}^{T}\mathit{PPpty}) : {}^{T}\mathit{PPpty}$

7.6 Summary of resources introduced

Items that are new since Chapter 6 are marked "**New!**" and items that have been revised since Chapter 6 are marked "**Revised!**".

7.6.1 Universal grammar resources

7.6.1.1 Types

$$\mathit{Loc} \ \text{—} \ \begin{bmatrix} \text{x-coord} & : & \mathit{Real} \\ \text{y-coord} & : & \mathit{Real} \\ \text{z-coord} & : & \mathit{Real} \end{bmatrix}$$

Phon — a basic type

e : *Phon* iff e is a phonological event

$$\mathit{SEvent} \ \text{—} \ \begin{bmatrix} \text{e-loc} & : & \mathit{Loc} \\ \text{sp} & : & \mathit{Ind} \\ \text{au} & : & \mathit{Ind} \\ \text{e} & : & \mathit{Phon} \\ \text{c}_{\text{loc}} & : & \text{loc(e,e-loc)} \\ \text{c}_{\text{sp}} & : & \text{speaker(e,sp)} \\ \text{c}_{\text{au}} & : & \text{audience(e,au)} \end{bmatrix} \ \text{(as in Chapter 2)}$$

Assgnmnt **Revised!** — a basic type

r : *Assgnmnt* iff r : *Rec* and $\text{labels}(r) \subset \{\text{x}_0, \text{x}_1, \ldots\}$

If T is *Assgnmnt*$\wedge T'$, T' is a record type and $\ell \in \text{labels}(T')$, then

1. if $\text{labels}(T') = \{\ell\}$, $T \ominus \ell = \textit{Assgnmnt}$
2. otherwise, $T \ominus \ell = \textit{Assgnmnt} \wedge (T' \ominus \ell/T)$

PropCntxt **Revised!** — a basic type

r : *PropCntxt* iff r : *Rec* and $\text{labels}(r) \cap \{\text{x}_0, \text{x}_1, \ldots\} = \emptyset$

If T is *PropCntxt*$\wedge T'$ and $\pi \in \text{tpaths}(T')$ then

1. if π is ℓ and $\text{labels}(T') = \{\ell\}$, then $T \ominus \pi = \textit{PropCntxt}$
2. otherwise, $T \ominus \pi = \textit{PropCntxt} \wedge (T' \ominus \pi/T')$

Cntxt **Revised!** — $\left[\begin{array}{lll} \mathfrak{s} & : & \textit{Assgnmnt} \\ \mathfrak{w} & : & \textit{Assgnmnt} \\ \mathfrak{g} & : & \textit{Assgnmnt} \\ \mathfrak{c} & : & \textit{PropCntxt} \end{array}\right]$

CntxtType — a basic type

T : *CntxtType* iff $T \sqsubseteq \textit{Cntxt}$

xType — a basic type

T : *xType* iff T : *RecType* and x $\in \text{labels}(T)$

Ppty — $\left[\begin{array}{lll} \text{bg} & : & \textit{xType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{RecType}) \end{array}\right]$

purification of properties, $\mathcal{P}(P)$ New!

If P : *Ppty*, then

if $P.\text{bg}^x = P.\text{bg}$, then

$\mathfrak{P}(P) = P$

otherwise:

$\mathfrak{P}(P)$ is $\ulcorner\lambda r{:}P.\text{bg}^{\text{x}}\ .\ \left[\begin{array}{lll} \mathfrak{c} & : & P.\text{bg} \parallel \left[\text{x}{=}r.\text{x}\right] \\ \text{e} & : & P(\mathfrak{c}) \end{array}\right]\urcorner$

purification$^\forall$ of properties, $\mathcal{P}^\forall(P)$ New!

If P : *Ppty*, then

if $P.\text{bg}^x = P.\text{bg}$, then

$\mathfrak{P}^\forall(P) = P$

otherwise:

$\mathfrak{P}^{\forall}(P)$ is $\ulcorner\lambda r{:}P.\text{bg}^{\text{x}}\ .\ ((r'{:}P.\text{bg}\,\|\big[\text{x}{=}r.\text{x}\big]) \to \big[\ \text{e} \quad : \quad P(r')\ \ \big])\urcorner$

$P\{a\}$ **New!**

If P is a pure property, $P\{a\}$ represents the type $P(\big[\text{x}{=}a\big])$

$\mathfrak{T}(P)$ **New!**

If P : *Ppty* and P is pure, then $\mathfrak{T}(P)$: *Type*.

a : $\mathfrak{T}(P)$ iff $\mathfrak{P}(P)\{a\}$ is witnessed.

exist$^{\text{w}}(P)$ **New!**

If P : *Ppty*, then exist$^{\text{w}}(P)$: *Type*.

X : exist$^{\text{w}}(P)$ iff

1. X : $\text{set}(\mathfrak{T}(P))$
2. $|X| = 1$
   (equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) = \frac{1}{|[\check{\ }\mathfrak{T}(P)]|}$)

exist$^{\text{w}}_{\text{pl}}(P)$ **New!**

If P : *Ppty*, then exist$^{\text{w}}_{\text{pl}}(P)$: *Type*.

X : exist$^{\text{w}}_{\text{pl}}(P)$ iff

1. X : $\text{set}(\mathfrak{T}(P))$
2. $|X| \geq 2$
   (equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \frac{2}{|[\check{\ }\mathfrak{T}(P)]|}$)

no$^{\text{w}}(P)$ **New!**

If P : *Ppty*, then no$^{\text{w}}(P)$: *Type*.

X : no$^{\text{w}}(P)$ iff

1. X : $\text{set}(\mathfrak{T}(P))$
2. $|X| = 0$
   (equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) = 0$)

equivalently,

X : no$^{\text{w}}(P)$ iff $X = \emptyset$

every$^{\text{w}}(P)$ **New!**

If P : *Ppty*, then every$^{\text{w}}(P)$: *Type*.

X : every$^{\text{w}}(P)$ iff

1. X : $\text{set}(\mathfrak{T}(P))$
2. $|X| = |[\check{\ }\mathfrak{T}(P)]|$
   (equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) = 1$)

equivalently,

X : every$^{\text{w}}(P)$ iff $X = [\check{\ }\mathfrak{T}(P)]$

$\text{most}^{\text{w}}(P)$ **New!**

If P : *Ppty*, then $\text{most}^{\text{w}}(P)$: *Type*.

X : $\text{most}^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\text{most}}(P)$, where $.5 < \theta_{\text{most}}(P) < 1$ (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \geq \theta_{\text{most}}(P)$)

$\text{many}_a^{\text{w}}(P)$ **New!**

If P : *Ppty*, then $\text{many}_a^{\text{w}}(P)$: *Type*.

X : $\text{many}_a^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \geq \theta_{\text{many}_a}(P)$, where $\theta_{\text{many}_a}(P)$ is a natural number, i, such that $i > 2$. (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \geq \frac{\theta_{\text{many}_a}(P)}{[[\check{\mathfrak{T}}(P)]]}$)

$\text{many}_p^{\text{w}}(P)$ **New!**

If P : *Ppty*, then $\text{many}_p^{\text{w}}(P)$: *Type*.

X : $\text{many}_p^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\text{many}_p}(P)$, where $0 < \theta_{\text{many}_p}(P) < 1$ (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \geq \theta_{\text{many}_p}(P)$)

$\text{few}_a^{\text{w}}(P)$ **New!**

If P : *Ppty*, then $\text{few}_a^{\text{w}}(P)$: *Type*.

X : $\text{few}_a^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \leq \theta_{\text{few}_a}(P)$, where $\theta_{\text{few}_a}(P)$ is a natural number, i, such that $i > 2$ (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \leq \frac{\theta_{\text{few}_a}(P)}{[[\check{\mathfrak{T}}(P)]]}$)

$\text{few}_p^{\text{w}}(P)$ **New!**

If P : *Ppty*, then $\text{few}_p^{\text{w}}(P)$: *Type*.

X : $\text{few}_p^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\downarrow P]|} \leq \theta_{\text{few}_p}(P)$, where $0 < \theta_{\text{few}_p}(P) < 1$ (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \leq \theta_{\text{few}_p}(P)$)

$\text{a\_few}_a^{\text{w}}(P)$ **New!**

If P : *Ppty*, then $\text{a\_few}_a^{\text{w}}(P)$: *Type*.

X : $\text{a\_few}_a^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \geq \theta_{\text{few}_a}(P)$, where $\theta_{\text{few}_a}(P)$ is a natural number, i, such that $i > 2$ (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \geq \frac{\theta_{\text{few}_a}(P)}{[[\check{\mathfrak{T}}(P)]]}$)

a_few$^{\text{w}}_{p}(P)$ **New!**

If P : *Ppty*, then a_few$^{\text{w}}_{p}(P)$: *Type*.

X : a_few$^{\text{w}}_{p}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\text{few}_p}(P)$, where $0 < \theta_{\text{few}_p}(P) < 1$
   (equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \theta_{\text{few}_p}(P)$)

^{T}*Ppty* **New!** — if T is a type, then ^{T}*Ppty* is a type

P : ^{T}*Ppty* iff P : *Ppty* and P.bg $\sqsubseteq \left[\text{x}:T\right]$

PlPpty — a basic type

P : *PlPpty* iff P : *Ppty* and for some type T, P.bg $\sqsubseteq \left[\text{x}:\text{plurality}(T)\right]$

PPpty — $\left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{Ppty}) \end{array}\right]$

^{T}*PPpty* **New!** — if T is a type, then ^{T}*PPpty* is a type

$\mathcal{P}$: ^{T}*PPpty* iff $\mathcal{P}$: *PPpty* and for any c : $\mathcal{P}$.bg, $\mathcal{P}(c)$: ^{T}*Ppty*

Quant — (*Ppty*→*RecType*)

PQuant — $\left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{Quant}) \end{array}\right]$

QuantDet — (*Ppty*→*Quant*)

PQuantDet — $\left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{QuantDet}) \end{array}\right]$

PRecType — $\left[\begin{array}{lll} \text{bg} & : & \textit{CntxtType} \\ \text{fg} & : & (\text{bg}\rightarrow\textit{RecType}) \end{array}\right]$

Cont — *PRecType*∨*PPpty*∨*PQuant*∨*PQuantDet*

Cat — a basic type

s, np, det, n, v, vp : *Cat*

Syn — $\left[\begin{array}{lll} \text{cat} & : & \textit{Cat} \\ \text{daughters} & : & \textit{Sign}^* \end{array}\right]$

Sign — a basic type

σ : *Sign* iff σ : $\left[\begin{array}{lll} \text{s-event} & : & \textit{SEvent} \\ \text{syn} & : & \textit{Syn} \\ \text{cont} & : & \textit{Cont} \end{array}\right]$

SignType — a basic type

T : *SignType* iff $T \sqsubseteq$ *Sign*

S — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=s:}\textit{Cat}\end{bmatrix}\end{bmatrix}$)

S/i **New!** — if i is a natural number, then *S/i* is a type

α : S/i iff α : *S* and α.cont.bg $\sqsubseteq \begin{bmatrix}\mathfrak{g}\text{:}\begin{bmatrix}\text{x}_i\text{:}\textit{Ind}\end{bmatrix}\end{bmatrix}$

NP — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=np:}\textit{Cat}\end{bmatrix}\end{bmatrix}$

whNP **New!** — a basic type

σ : *WhNP* iff σ : *NP*, σ.cont is $\mathcal{Q}$ and $\mathcal{Q}$.bg $\sqsubseteq \begin{bmatrix}\mathfrak{w}\text{:}\begin{bmatrix}\text{x}_i\text{:}\textit{Ind}\end{bmatrix}\end{bmatrix}$, for some natural number i.

NP_{wh_i} **New!** — if i is a natural number, then NP_{wh_i} is a type

α : NP_{wh_i} iff α : *NP* and α.cont.bg $\sqsubseteq \begin{bmatrix}\mathfrak{w}\text{:}\begin{bmatrix}\text{x}_i\text{:}\textit{Ind}\end{bmatrix}\end{bmatrix}$

Det — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=det:}\textit{Cat}\end{bmatrix}\end{bmatrix}$

N — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=n:}\textit{Cat}\end{bmatrix}\end{bmatrix}$

${}^{T}N$ **New!** — if T is a type, then ${}^{T}N$ is a type

α : ${}^{T}N$ iff α : *N* and α.cont : ${}^{T}\textit{PPpty}$

V — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=v:}\textit{Cat}\end{bmatrix}\end{bmatrix}$

VP — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=vp:}\textit{Cat}\end{bmatrix}\end{bmatrix}$

Rel **New!** — $\begin{bmatrix}\textit{Sign}\\ \text{syn:}\begin{bmatrix}\text{cat=rel:}\textit{Cat}\end{bmatrix}\end{bmatrix}$

${}^{T}\textit{Rel}$ **New!** — if T is a type, then ${}^{T}\textit{Rel}$ is a type

α : ${}^{T}\textit{Rel}$ iff α : *Rel* and α.cont : ${}^{T}\textit{PPpty}$

NoDaughters — $\begin{bmatrix}\text{syn:}\begin{bmatrix}\text{daughters=}\varepsilon\text{:}\textit{Sign}^*\end{bmatrix}\end{bmatrix}$

Real — a basic type

n : *Real* iff n is a real number

Card — a basic type

n : *Card* iff n is a cardinal number (natural numbers with the addition of $\aleph_0, \aleph_1, \ldots$)

$$\mathit{AmbTempFrame} \;\text{—}\; \left[\begin{array}{lll} \text{x} & : & \mathit{Real} \\ \text{loc} & : & \mathit{Loc} \\ \text{e} & : & \text{temp(loc, x)} \end{array}\right]$$

$$\mathit{TempRiseEventCntxt} \;\text{—}\; \left[\begin{array}{lll} \text{fix} & : & \left[\begin{array}{lll} \text{loc} & : & \mathit{Loc} \end{array}\right] \\ \text{scale} & : & (\mathit{AmbTempFrame} \rightarrow \mathit{Real}) \end{array}\right]$$

TempRiseEvent —

$$\lambda r{:}\mathit{TempRiseEventCntxt}\;.\; \left[\begin{array}{lll} \text{e} & : & (\mathit{AmbTempFrame}\|r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale(e[0])} < r.\text{scale(e[1])} \end{array}\right]$$

$$\mathit{PriceFrame} \;\text{—}\; \left[\begin{array}{lll} \text{x} & : & \mathit{Real} \\ \text{loc} & : & \mathit{Loc} \\ \text{commodity} & : & \mathit{Ind} \\ \text{e} & : & \text{price(commodity, loc, x)} \end{array}\right]$$

$$\mathit{PriceRiseEventCntxt} \;\text{—}\; \left[\begin{array}{lll} \text{fix} & : & \left[\begin{array}{lll} \text{loc} & : & \mathit{Loc} \\ \text{commodity} & : & \mathit{Ind} \end{array}\right] \\ \text{scale} & : & (\mathit{PriceFrame} \rightarrow \mathit{Real}) \end{array}\right]$$

PriceRiseEvent —

$$\lambda r{:}\mathit{TempRiseEventCntxt}\;.\; \left[\begin{array}{lll} \text{e} & : & (\mathit{PriceFrame}\|r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale(e[0])} < r.\text{scale(e[1])} \end{array}\right]$$

$$\mathit{LocFrame} \;\text{—}\; \left[\begin{array}{lll} \text{x} & : & \mathit{Ind} \\ \text{loc} & : & \mathit{Loc} \\ \text{e} & : & \text{at(x, loc)} \end{array}\right]$$

$$\mathit{LocRiseEventCntxt} \;\text{—}\; \left[\begin{array}{lll} \text{fix} & : & \left[\begin{array}{lll} \text{x} & : & \mathit{Ind} \end{array}\right] \\ \text{scale} & : & (\mathit{LocFrame} \rightarrow \mathit{Real}) \end{array}\right]$$

LocRiseEvent —

$$\lambda r{:}\mathit{LocRiseEventCntxt}\;.\; \left[\begin{array}{lll} \text{e} & : & (\mathit{LocFrame}\|r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale(e[0])} < r.\text{scale(e[1])} \end{array}\right]$$

Topos — a basic type

$$\text{If } \tau : \mathit{Topos}\text{, then } \tau : \left[\begin{array}{lll} \text{bg} & : & \mathit{Type} \\ \text{fg} & : & (\text{bg}\rightarrow\mathit{Type}) \end{array}\right]$$

7.6.1.2 Predicates

with arity $\langle$*Phon*, *Loc*$\rangle$

loc — e : loc(u, l) iff u is located at l in e

with arity $\langle$*Phon*, *Ind*$\rangle$

speaker — e : speaker(u, a) iff u is the speaker of u in e

audience — e : audience(u, a) iff u is the audience of u in e

with arity $\langle$*Card*$\rangle$

card — X : card(n) iff for some T, $X : \mathrm{set}(T)$ and $|X| = n$

card_at_least — X : card_at_least(n) iff for some T, $X : \mathrm{set}(T)$ and $|X| \geq n$

card_at_most — X : card_at_most(n) iff for some T, $X : \mathrm{set}(T)$ and $|X| \leq n$

with arity $\langle$*Ppty*$\rangle$

unique — s : unique(P) iff $|\,[{\downarrow}P \restriction s]\,| = 1$

with arity $\langle$*Ppty*, *Ppty*$\rangle$

exist **Revised!**

general witness condition

$$s : \text{exist}(P, Q) \text{ iff } s : \left[\begin{array}{lll} \text{X} & : & \text{exist}^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{array}\right]$$

particular witness condition

$$s : \text{exist}(P, Q) \text{ iff } s : \left[\begin{array}{lll} \text{x} & : & \mathfrak{T}(P) \\ \text{e} & : & \mathfrak{P}(Q)\{\text{x}\} \end{array}\right]$$

exist_{pl} **New!**

general witness condition

$$s : \text{exist}_{\text{pl}}(P, Q) \text{ iff } s : \left[\begin{array}{lll} \text{X} & : & \text{exist}^w_{\text{pl}}(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \rightarrow \mathfrak{P}(Q)\{a\}) \end{array}\right]$$

no **New!**

general witness condition

$$s : \text{no}(P, Q) \text{ iff } s : \left[\begin{array}{lll} \text{X} & : & \text{no}^w(P) \\ \text{f} & : & ((a : (\mathfrak{T}(P) \wedge \mathfrak{T}(Q))) \rightarrow \left[\begin{array}{lll} \text{x=}a & : & \mathfrak{T}(\text{X}) \end{array}\right]) \end{array}\right]$$

particular witness condition

$$s : \text{no}(P, Q) \text{ iff } s : \left[\begin{array}{lll} \text{X} & : & \text{every}^w(P) \\ \text{f} & : & ((x : \mathfrak{T}(X)) \rightarrow \neg\mathfrak{P}(Q)\{x\}) \end{array}\right]$$

every **Revised!**

general witness condition

$$s : \text{every}(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{every}^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \to \mathfrak{P}(Q)\{a\}) \end{bmatrix}$$

most **New!**

general witness condition

$$s : \text{most}(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{most}^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \to \mathfrak{P}(Q)\{a\}) \end{bmatrix}$$

many$_a$ **New!**

general witness condition

$$s : \text{many}_a(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{many}_a^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \to \mathfrak{P}(Q)\{a\}) \end{bmatrix}$$

many$_p$ **New!**

general witness condition

$$s : \text{many}_p(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{many}_p^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \to \mathfrak{P}(Q)\{a\}) \end{bmatrix}$$

few$_a$ **New!**

general witness condition

$$s : \text{few}_a(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{few}_a^w(P) \\ \text{f} & : & ((a : (\mathfrak{T}(P) \wedge \mathfrak{T}(Q))) \to \begin{bmatrix} \text{x}{=}a & : & \mathfrak{T}(\text{X}) \end{bmatrix}) \end{bmatrix}$$

particular witness condition

$$s : \text{few}_a(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \overline{\text{few}_a^w(P)} \\ \text{f} & : & ((x : \mathfrak{T}(X)) \to \neg\mathfrak{P}(Q)\{x\}) \end{bmatrix}$$

few$_p$ **New!**

general witness condition

$$s : \text{few}_p(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{few}_p^w(P) \\ \text{f} & : & ((a : (\mathfrak{T}(P) \wedge \mathfrak{T}(Q))) \to \begin{bmatrix} \text{x}{=}a & : & \mathfrak{T}(\text{X}) \end{bmatrix}) \end{bmatrix}$$

particular witness condition

$$s : \text{few}_p(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \overline{\text{few}_p^w(P)} \\ \text{f} & : & ((x : \mathfrak{T}(X)) \to \neg\mathfrak{P}(Q)\{x\}) \end{bmatrix}$$

a_few$_a$ **New!**

general witness condition

$$s : \text{a\_few}_a(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{a\_few}_a^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \to \mathfrak{P}(Q)\{a\}) \end{bmatrix}$$

a_few$_p$ **New!**

general witness condition

$$s : \text{a\_few}_p(P, Q) \text{ iff } s : \begin{bmatrix} \text{X} & : & \text{a\_few}_p^w(P) \\ \text{f} & : & ((a : \mathfrak{T}(\text{X})) \to \mathfrak{P}(Q)\{a\}) \end{bmatrix}$$

with arity $\langle \mathit{PlPpty}, \mathit{PlPpty} \rangle$

exactly_n — for n a natural number,
s : exactly_$n(P, Q)$ iff s : at_least_$n(P, Q) \wedge$at_most_$n(P, Q)$

at_least_n — for n a natural number,
s : at_least_$n(P, Q)$ iff $[\check{\mathcal{F}}((Q \upharpoonright s).\text{fg} \mid_{\mathcal{F}(P.\text{fg})}) \dot{\wedge} \left[\text{x:card\_at\_least}(n)\right]] \neq \emptyset$

at_most_n — for n a natural number,
s : at_most_$n(P, Q)$ iff $r : \mathcal{F}((Q \upharpoonright s).\text{fg} \mid_{\mathcal{F}(P.\text{fg})})$ implies r : $\left[\text{x:card\_at\_most}(n)\right]$

with arity $\{\langle T \rangle \mid T \text{ is a type}\}$

be — e : be(a) iff $a \varepsilon e$

with arity $\langle \mathit{Loc}, \mathit{Real} \rangle$

temp — e : temp(l, n) iff n is the temperature at l in e.

with arity $\langle \mathit{Real}, \mathit{Real} \rangle$

less-than — e : less-than(n, m) iff $n \varepsilon e$, $m \varepsilon e$ and $n < m$

with arity $\langle \mathit{Type}, \mathit{Type}, \mathit{Topos} \rangle$

nec —
If $\mathbb{T}$ is a modal type system and $p \in \mathbb{T}$, then
$s :_p \text{nec}(T, B, \tau)$ iff $s :_p B$, $B \sqsubseteq_{\mathbb{T}} \tau.\text{bg}$ and $\tau(s) \sqsubseteq_{\mathbb{T}} T$

poss —
If $\mathbb{T}$ is a modal type system and $p \in \mathbb{T}$, then
$s :_p \text{poss}(T, B, \tau)$ iff $s :_p B$, $B \sqsubseteq_{\mathbb{T}} \tau.\text{bg}$ and $\tau(s) \top_{\mathbb{T}} T$

with arity $\langle \mathit{RecType}, \mathit{RecType} \rangle$

pov — e : pov(T_1, T_2) iff T_2 is a point of view on T_1 in e.
e : pov(T_1, T_2) implies $\text{labels}(T_2) \subseteq \text{labels}(\text{T}_1)$

with arity $\langle \mathit{Ind}, \mathit{RecType} \rangle$

ltm — e : ltm(a, T) iff T is a's long term memory in e.

rbelieve — e : rbelieve(a, T) iff T is a's religious beliefs in e.

des — e : des(a, T) iff T is a's desires in e.

7.6.1.3 Properties

$P_1 \& P_2$ **New!**

If T is a type, $P_1 : {}^{T}\!Ppty$ and $P_2 : {}^{T}\!Ppty$, then *the conjunction of* P_1 *and* P_2, $P_1 \& P_2$, is

$$\ulcorner \lambda r{:}\big[\mathrm{x}{:}T\big] \;.\; \begin{bmatrix} \mathrm{e}_1 & : & P_1\{r.\mathrm{x}\} \\ \mathrm{e}_2 & : & P_2\{r.\mathrm{x}\} \end{bmatrix} \urcorner$$

7.6.1.4 Scales

(as in Chapter 5)

7.6.1.5 Lexicon

Lex

If T_{phon} is a phonological type (that is, $T_{\text{phon}} \sqsubseteq$ *Phon*) and T_{sign} is a sign type (that is, $T_{\text{sign}} \sqsubseteq$ *Sign*), then we shall use Lex(T_{phon}, T_{sign}) to represent

$$((T_{\text{sign}} \mathbin{\dot{\wedge}} \big[\text{s-event:}\big[\text{e:}T_{\text{phon}}\big]\big]) \mathbin{\dot{\wedge}} \mathit{NoDaughters})$$

SemCommonNoun(T_{bg}, p)

If p is a predicate with arity $\langle$*Ind*$\rangle$ and T_{bg} is a type (of context), then SemCommonNoun(T_{bg}, p) is

$$\ulcorner \lambda c{:}T_{\text{bg}} \;.\; \ulcorner \lambda r{:}\big[\mathrm{x}{:}\mathit{Ind}\big] \;.\; \big[\;\mathrm{e} \;\;:\;\; p(r.\mathrm{x})\;\big] \urcorner\urcorner$$

If p is a predicate with arity $\langle$*Rec*$\rangle$ and T_{bg} is a type (of context), then SemCommonNoun(T_{bg}, p) is

$$\ulcorner \lambda c{:}T_{\text{bg}} \;.\; \ulcorner \lambda r{:}\big[\mathrm{x}{:}\mathit{Rec}\big] \;.\; \big[\;\mathrm{e} \;\;:\;\; p(r.\mathrm{x})\;\big] \urcorner\urcorner$$

$\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$

If T_{phon} is a phonological type, p is a predicate with arity $\langle$*Ind*$\rangle$ or $\langle$*Rec*$\rangle$ and T_{bg} is a type (of context), then $\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$ is

$$\text{Lex}(T_{\text{phon}}, N) \mathbin{\dot{\wedge}} \big[\text{cont=SemCommonNoun}(T_{\text{bg}}, p)\text{:}\mathit{PPpty}\big]$$

SemPropName(T_{phon})

If T_{phon} is a phonological type, then SemPropName(T_{phon}) is

$$\ulcorner \lambda c{:}\begin{bmatrix} \mathit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix} \mathrm{x}{:}\mathit{Ind} \\ \mathrm{e}{:}\text{named}(\mathrm{x}, T_{\text{phon}}) \end{bmatrix} \end{bmatrix} \;.\; \lambda P{:}\mathit{Ppty} \;.\; P(c.\mathfrak{c}) \urcorner$$

$\text{Lex}_{\text{PropName}}(T_{\text{phon}})$

If T_{phon} is a phonological type,

then $\text{Lex}_{\text{PropName}}(T_{\text{phon}})$ is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot{\wedge}} \left[\text{cnt=SemPropName}(T_{\text{phon}}):\textit{PQuant}\right]$$

SemPron

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}{:}\left[\text{x}_0{:}\textit{Ind}\right]\end{bmatrix} \,.\, \lambda P{:}\textit{Ppty} \,.\, P(\left[\text{x=}c.\mathfrak{s}.\text{x}_0\right])\urcorner$$

$\text{LexPron}(T_{\text{phon}})$

If T_{phon} is a phonological type, then $\text{LexPron}(T_{\text{phon}})$ is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot{\wedge}} \left[\text{cont=SemPron}:\textit{PQuant}\right]$$

SemWhPron **New!**

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{w}{:}\left[\text{x}_0{:}\textit{Ind}\right]\end{bmatrix} \,.\, \lambda P{:}\textit{Ppty} \,.\, P(\left[\text{x=}c.\mathfrak{w}.\text{x}_0\right])\urcorner$$

$\text{LexWhPron}(T_{\text{phon}})$ **New!**

If T_{phon} is a phonological type, then $\text{LexWhPron}(T_{\text{phon}})$ is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot{\wedge}} \left[\text{cont=SemWhPron}:\textit{PQuant}\right]$$

$\text{SemNumeral}(n)$

If n is a real number, then $\text{SemNumeral}(n)$ is

$$\ulcorner\lambda c{:}\textit{Cntxt} \,.\, \lambda P{:}\textit{Ppty} \,.\, P(\left[\text{x=}n\right])\urcorner$$

$\text{Lex}_{\text{numeral}}(T_{\text{phon}}, n)$

If T_{phon} is a phonological type and n is a real number, then $\text{Lex}_{\text{numeral}}(T_{\text{phon}}, n)$ is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot{\wedge}} \left[\text{cnt=SemNumeral}(n):\textit{PQuant}\right]$$

SemIndefArt

$$\begin{array}{l}\lambda Q{:}\textit{Ppty} \,.\\ \quad\ulcorner\lambda c{:}\textit{Cntxt} \,.\\ \quad\quad\lambda P{:}\textit{Ppty} \,.\\ \quad\quad\quad\begin{bmatrix}\text{restr=}Q & : & \textit{Ppty}\\ \text{scope=}P|_{\mathfrak{F}(\text{restr})} & : & \textit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{bmatrix}\urcorner\end{array}$$

$\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \textit{Det}) \mathbin{\dot{\wedge}} \left[\text{cont=SemIndefArt:}(\textit{Ppty}\rightarrow\textit{PQuant})\right]$$

SemUniversal

$$\begin{array}{l}\lambda Q{:}\textit{Ppty}\ .\\ \ \ulcorner\lambda c{:}\textit{Cntxt}\ .\\ \quad\lambda P{:}\textit{Ppty}\ .\\ \qquad\left[\begin{array}{lll}\text{restr}=Q & : & \textit{Ppty}\\ \text{scope}=P|_{\mathfrak{F}(\text{restr})} & : & \textit{Ppty}\\ \text{e} & : & \text{every(restr, scope)}\end{array}\right]\urcorner\end{array}$$

$\text{Lex}_{\text{Universal}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{Universal}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \textit{Det}) \mathbin{\dot{\wedge}} \left[\text{cont=SemUniversal:}(\textit{Ppty}\rightarrow\textit{PQuant})\right]$$

SemDefArt

$$\begin{array}{l}\lambda Q{:}\textit{Ppty}\ .\\ \ \ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{c}{:}\left[\text{e:unique}(Q)\right]\end{array}\right]\ .\\ \quad\lambda P{:}\textit{Ppty}\ .\\ \qquad\left[\begin{array}{lll}\text{restr}=Q\upharpoonright c.\mathfrak{c}.\text{e} & : & \textit{Ppty}\\ \text{scope}=P|_{\mathfrak{F}(\text{restr})} & : & \textit{Ppty}\\ \text{e} & : & \text{every(restr, scope)}\end{array}\right]\urcorner\end{array}$$

$\text{Lex}_{\text{DefArt}}(T_{\text{Phon}})$

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \textit{Det}) \mathbin{\dot{\wedge}} \left[\text{cont=SemDefArt:}(\textit{Ppty}\rightarrow\textit{PQuant})\right]$$

SemIntransVerb(T_{bg}, p)

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \textit{Ind}\rangle$, then SemIntransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\left[\text{x:}\textit{Ind}\right]\ .\ \left[\ \text{e}\ :\ p(r.\text{x})\ \right]\urcorner\urcorner$$

If $T_{\text{bg}} \sqsubseteq \left[\mathfrak{c}{:}\textit{Rec}\right]$ is a record type (for context) and p is a predicate with arity $\langle \textit{Rec}, \textit{Rec}\rangle$, then SemIntransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\left[\text{x:}\textit{Rec}\right]\ .\ \left[\ \text{e}\ :\ p(r.\text{x}, c.\mathfrak{c})\ \right]\urcorner\urcorner$$

$\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$

If T_{phon} is a phonological type, $T_{\text{bg}} \sqsubseteq \left[\mathfrak{c}{:}\textit{Rec}\right]$ a record type (for context) and p is a predicate with arity $\langle \textit{Ind}\rangle$ or $\langle \textit{Rec}, \textit{Rec}\rangle$, then $\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$ is

$$\text{Lex}(T_{\text{phon}}, V_i) \mathbin{\dot{\wedge}} \left[\text{cnt=SemIntransVerb}(T_{\text{bg}}, p){:}\textit{PPpty}\right]$$

SemTransVerb(T_{bg}, p)

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle Ind, Ind\rangle$, then SemTransVerb(T_{bg}, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \lambda\mathcal{Q}{:}Quant\ .\ \ulcorner\lambda r_1{:}\big[\text{x}{:}Ind\big]\ .\ \mathcal{Q}(\ulcorner\lambda r_2{:}\big[\text{x}{:}Ind\big]\ .\ \big[\ \text{e}\ :\ p(r_1.\text{x}, r_2.\text{x})\ \big]\urcorner)\urcorner\urcorner$$

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle Ind, Quant\rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \lambda\mathcal{Q}{:}Quant\ .\ \ulcorner\lambda r{:}\big[\text{x}{:}Ind\big]\ .\ \big[\ \text{e}\ :\ p(r.\text{x}, \mathcal{Q})\ \big]\urcorner)\urcorner$$

Lex$_{\text{TransVerb}}$(T_{phon}, T_{bg}, p)

If T_{phon} is a phonological type, T_{bg} a record type (for context) and p is a predicate with arity $\langle Ind, Ind\rangle$ or $\langle Ind, Quant\rangle$, then Lex$_{\text{TransVerb}}$($T_{\text{phon}}$, T_{bg}, p) is

$$\text{Lex}(T_{\text{phon}}, V_t)\ \dot\wedge\ \big[\text{cnt}{=}\text{SemTransVerb}(T_{\text{bg}}, p){:}PPpty\big]$$

SemBe

SemBe$_{\text{ID}}$

$$\begin{array}{l}\ulcorner\lambda c{:}\begin{bmatrix}Cntxt\\ \mathfrak{c}{:}\big[\text{ty}{:}Type\big]\end{bmatrix}\ .\\ \quad\lambda\mathcal{Q}{:}Quant\ .\\ \quad\quad\ulcorner\lambda r_1{:}\big[\text{x}{:}c.\mathfrak{c}.\text{ty}\big]\ .\\ \quad\quad\quad\mathcal{Q}(\ulcorner\lambda r_2{:}\big[\text{x}{:}c.\mathfrak{c}.\text{ty}\big]\ .\ \begin{bmatrix}\text{x}{=}r_1.\text{x}, r_2.\text{x} & : & c.\mathfrak{c}.\text{ty}\\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner)\urcorner\urcorner\end{array}$$

SemBe$_{\text{scalar}}$

$$\begin{array}{l}\ulcorner\lambda c{:}\begin{bmatrix}Cntxt\\ \mathfrak{c}{:}\begin{bmatrix}\text{ty}{:}Type\\ \text{sc}{:}(\text{ty}\rightarrow Real)\end{bmatrix}\end{bmatrix}\ .\\ \quad\lambda\mathcal{Q}{:}Quant\ .\\ \quad\quad\ulcorner\lambda r_1{:}\big[\text{x}{:}c.\mathfrak{c}.\text{ty}\big]\ .\\ \quad\quad\quad\mathcal{Q}(\ulcorner\lambda r_2{:}\big[\text{x}{:}Real\big]\ .\ \begin{bmatrix}\text{x}{=}c.\mathfrak{c}.\text{sc}(r_1.\text{x}), r_2.\text{x} & : & Real\\ \text{e} & : & \text{be(x)}\end{bmatrix}\urcorner)\urcorner\urcorner\end{array}$$

Lex$_{\text{be}}$(T_{Phon})

If T_{Phon} is a phonological type, then Lex$_{\text{be}_{\text{ID}}}$(T_{Phon}) is

$$\text{Lex}(T_{\text{Phon}}, V)\ \dot\wedge\ \big[\text{cont}{=}\text{SemBe}_{\text{ID}}{:}(Quant\rightarrow Ppty)\big]$$

If T_{Phon} is a phonological type, then Lex$_{\text{be}_{\text{scalar}}}$($T_{\text{Phon}}$) is

$$\text{Lex}(T_{\text{Phon}}, V)\ \dot\wedge\ \big[\text{cont}{=}\text{SemBe}_{\text{scalar}}{:}(Quant\rightarrow Ppty)\big]$$

$\mathrm{FrameType}(p)$

$\mathrm{FrameType}$ is a partial function on predicates, p, with arity $\langle$*Ind*$\rangle$ which can be defined for particular agents and particular times, which obeys the constraint:

$$\mathrm{FrameType}(p) \sqsubseteq \begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{e} & : & p(\mathrm{x}) \end{bmatrix}$$

p_frame

1. If p is a predicate in the domain of $\mathrm{FrameType}$, then p_frame is a predicate with arity $\langle$*Rec*$\rangle$.
2. $e : p\text{\_frame}(r)$ iff $r : \mathrm{FrameType}(p)$ and $e = r$

p_pl

1. If p is a singular predicate (i.e. there is no p' such that $p = p'$_pl) with arity $\langle T\rangle$, then p_pl is a predicate with arity $\langle\mathrm{plurality}(T)\rangle$
2. $e : p\text{\_pl}(A)$ if for all $a \in A$, $e : p(a)$

CommonNounIndToFrame

If T_{phon} is a phonological type, p is a predicate with arity $\langle$*Ind*$\rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then
CommonNounIndToFrame(Lex$_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$$\mathrm{Lex}_{\mathrm{CommonNoun}}(T_{\mathrm{phon}}, T_{\mathrm{bg}}, p\text{\_frame})$$

RestrictCommonNoun

If T_{phon} is a phonological type, p is a predicate, T_{bg} and T_{res} are record types and Σ is Lex$_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$, then RestrictCommonNoun(Σ, T_{res}) is

$$\Sigma \boxed{\wedge} \left[\ \mathrm{cont}{=}\ulcorner\lambda c{:}T_{\mathrm{bg}}\ .\ \ulcorner\mathrm{SemCommonNoun}(T_{\mathrm{bg}}, p)(c)\,|_{T_{\mathrm{res}}}\urcorner\urcorner\ \ :\ \ \mathit{PPpty}\ \right]$$

IntransVerbIndToFrame

If T_{phon} is a phonological type, p is a predicate with arity $\langle$*Ind*$\rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then
IntransVerbIndToFrame(Lex$_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$$\mathrm{Lex}_{\mathrm{IntransVerb}}(T_{\mathrm{phon}}, T_{\mathrm{bg}}, p\text{\_frame})$$

PluralCommonNoun

We assume that 'pluralnoun' is a function that maps phonological types for singular common nouns to corresponding phonological types for plural common nouns.

If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T\rangle$ and T_{bg} is a record type then PluralCommonNoun(Lex$_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$\text{Lex}_{\text{CommonNoun}}(\text{pluralnoun}(T_{\text{phon}}), T_{\text{bg}}, p\_\text{pl})$

PluralIntransVerb

We assume that 'pluralverb' is a function that maps phonological types for singular verbs to corresponding phonological types for plural verbs.

If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T\rangle$ and T_{bg} is a record type then PluralIntransVerb($\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$\text{Lex}_{\text{IntransVerb}}(\text{pluralverb}(T_{\text{phon}}), T_{\text{bg}}, p\_\text{pl})$

TransVerbToVerbPhrase **New!**

If T_{phon} is a phonological type, T_{bg} a context type, p is a predicate with arity $\langle \mathit{Ind}, \mathit{Ind}\rangle$ or $\langle \mathit{Ind}, \mathit{Quant}\rangle$ and Σ is $\text{Lex}_{\text{TransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$, then TransVerbToVerbPhrase(Σ) is

$$\Sigma\ \boxed{\dot\wedge} \left[\begin{array}{lll} \text{cat=vp} & : & \mathit{Cat} \\ \text{cont=}\varphi & : & \mathit{PPpty} \end{array}\right]$$

where φ is

$$\ulcorner\lambda c{:}\Sigma.\text{cont.bg}\dot\wedge\left[\mathfrak{g}{:}\left[\text{x}_0{:}\mathit{Ind}\right]\right] .\ \Sigma.\text{cont}(c)(\lambda P{:}\mathit{Ppty}\ .\ P\{c.\mathfrak{g}.\text{x}_0\})\urcorner$$

7.6.1.6 Constituent structure

RuleDaughters($T_{\text{daughters}}, T_{\text{mother}}$)

If T_{mother} is a sign type and $T_{\text{daughters}}$ is a type of strings of signs then

RuleDaughters($T_{\text{daughters}}, T_{\text{mother}}$)

is

$$\lambda u{:}T_{\text{daughters}}\ .\ T_{\text{mother}} \dot\wedge \left[\text{syn:}\left[\text{daughters=}u{:}T_{\text{daughters}}\right]\right]$$

ConcatPhon

$$\lambda u{:}\left[\text{s-event:}\left[\text{e:}\mathit{Phon}\right]\right]^{+}\ .\ \left[\begin{array}{lll} \text{s-event} & : & \left[\begin{array}{lll} \text{e=concat}_i(u[i].\text{s-event.e}) & : & \mathit{Phon} \end{array}\right] \end{array}\right]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$

If T_{mother} is a sign type and $T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$ are sign types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1} \ldots T_{\text{daughter}_n}$$

represents

$$\text{RuleDaughters}(T_{\text{mother}}, T_{\text{daughter}_1}{}^\frown \ldots {}^\frown T_{\text{daughter}_n})\ddot\wedge\text{ConcatPhon}$$

$\alpha@\beta$ **Revised!**

If $\alpha : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow(T_1 \rightarrow T_2))\end{bmatrix}$ and $\beta : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow T_1)\end{bmatrix}$ then the *combination of* α *and* β *based on functional application*, $\alpha@\beta$, is

$$\ulcorner\lambda c{:}[\alpha.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}} \wedge\!\!\!\!\cdot\ \text{incr}([\beta.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}, \alpha.\text{bg})\ .\\ [\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}(c)(\text{incr}([\beta.\text{fg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}, \alpha.\text{bg})(c))\urcorner$$

ContForwardApp(T_{arg}, T_{res})

If T_{arg} and T_{res} are types, then ContForwardApp(T_{arg}, T_{res}) is

$$\lambda u{:}\begin{bmatrix}\text{cont:}\begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow(T_{\text{arg}} \rightarrow T_{\text{res}}))\end{bmatrix}\end{bmatrix}^\frown\begin{bmatrix}\text{cont:}\begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow T_{\text{arg}})\end{bmatrix}\end{bmatrix}\ .\\ \begin{bmatrix}\text{cont=}u[0].\text{cont}@u[1].\text{cont:}\begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow T_{\text{res}})\end{bmatrix}\end{bmatrix}$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ T'_{\text{daughter}_1}(T'_{\text{daughter}_2} : T_{\text{arg}}) : T_{\text{res}}$

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and T_{arg} and T_{res} are content types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ T'_{\text{daughter}_1}(T'_{\text{daughter}_2} : T_{\text{arg}}) : T_{\text{res}}$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \wedge\!\!\!\!\cdot\ \text{ContForwardApp}(T_{\text{arg}}, T_{\text{res}})$$

$\alpha@_{\text{wh}_{i,j}}\beta$ **New!**

If

1. $\alpha : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow\mathit{Quant})\end{bmatrix}$,
2. $\beta : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow\mathit{RecType})\end{bmatrix}$,
3. $\alpha.\text{bg} \sqsubseteq \begin{bmatrix}\mathfrak{w}{:}\begin{bmatrix}\text{x}_i{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$ for some natural number, i, and
4. $\beta.\text{bg} \sqsubseteq \begin{bmatrix}\mathfrak{g}{:}\begin{bmatrix}\text{x}_j{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$ for some natural number, j,

then *the* wh$_{i,j}$*-combination of* α *and* β, $\alpha@_{\text{wh}_{i,j}}\beta$, is

$$\ulcorner\lambda c{:}([\alpha.\text{bg}\ominus\text{paths}_{\mathfrak{w}.\text{x}_i}(\alpha.\text{bg})]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}\wedge\!\!\!\!\cdot\\ \text{incr}([\beta.\text{bg} \ominus \text{paths}_{\mathfrak{g}.\text{x}_j}(\beta.\text{bg})]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.a}, \alpha.\text{bg}))\ .\\ \mathfrak{P}(\ulcorner\lambda r_1{:}[\alpha.\text{bg}^{\mathfrak{w}.\text{x}_i}]_{\mathfrak{w}.\text{x}_i\rightsquigarrow\text{x}}\ .\\ \alpha_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f},\mathfrak{w}.\text{x}_i\rightsquigarrow\text{x}}(c[r_1])\ (\mathfrak{P}(\ulcorner\lambda r_2{:}[\beta.\text{bg}^{\mathfrak{g}.\text{x}_j}]_{\mathfrak{g}.\text{x}_j\rightsquigarrow\text{x}}\ .\\ \text{incr}(\beta_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a},\mathfrak{g}.\text{x}_j\rightsquigarrow\text{x}}, \alpha.\text{bg})(c[r_2])\urcorner))\urcorner)\urcorner$$

ContForwardApp$_{@_{\text{wh}_{i,j}}}(T_{\text{fun}}, T_{\text{arg}}, T_{\text{res}})$ **New!**

If T_{fun}, T_{arg} and T_{res} are types such that if $\alpha : T_{\text{fun}}$ and $\beta : T_{\text{arg}}$, then $\alpha @_{\text{wh}_{i,j}} \beta$ is defined and of type T_{res}, then

$$\text{ContForwardApp}_{@_{\text{wh}_{i,j}}}(T_{\text{fun}}, T_{\text{arg}}, T_{\text{res}})$$

is

$$\lambda u{:}\big[\text{cont:}T_{\text{fun}}\big]^\frown\big[\text{cont:}T_{\text{arg}}\big]\ .\ \big[\text{cont=}u[0].\text{cont}@_{\text{wh}_{i,j}}u[1].\text{cont:}T_{\text{res}}\big]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \ \mid\ T'_{\text{daughter}_1}\big(@_{\text{wh}_{i,j}} T'_{\text{daughter}_2} : T_{\text{arg}}\big) : T_{\text{res}}$ **New!**

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types, $T_{\text{daughter}_1} \sqsubseteq \big[\text{cont=}c_1\text{:}\mathit{Cont}\big]$ where $c_1.\text{bg} \sqsubseteq \big[\mathfrak{w}{:}\big[\text{x}_i{:}\mathit{Ind}\big]\big]$, $T_{\text{daughter}_2} \sqsubseteq \big[\text{cont=}c_2\text{:}\mathit{Cont}\big]$ where $c_2.\text{bg} \sqsubseteq \big[\mathfrak{g}{:}\big[\text{x}_j{:}\mathit{Ind}\big]\big]$ and T_{arg} and T_{res} are content types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \ \mid\ T'_{\text{daughter}_1}\big(@_{\text{wh}_{i,j}} T'_{\text{daughter}_2} : T_{\text{arg}}\big) : T_{\text{res}}$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \underset{\therefore}{\wedge} \text{ContForwardApp}_{@_{\text{wh}_{i,j}}}(T_{\text{arg}}, T_{\text{res}})$$

$\alpha@@\beta$

If $\alpha : (T_1 \to \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\to T_2)\end{bmatrix})$ and $\beta : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\to T_1)\end{bmatrix}$ then the *combination of α and β based on functional application*, $\alpha@@\beta$, is

$$\ulcorner\lambda c{:}\begin{bmatrix} \mathfrak{c} & : & \begin{bmatrix} \text{s} & : & \beta.\text{bg} \\ \text{f} & : & \alpha(\beta(s)).\text{bg} \\ \text{a=s.}\mathfrak{c} & : & \mathit{PropCntxt} \end{bmatrix} \\ \mathfrak{s}\text{=}\mathfrak{c}.\text{s}.\mathfrak{s} & : & \mathit{Assgnmnt} \end{bmatrix}\ .$$

$$[\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}([\beta]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}(c))(c)\urcorner$$

ContForwardApp$_{@@}(T_{\text{arg}}, T_{\text{res}})$

If T_{arg} and T_{res} are types, then ContForwardApp$_{@@}(T_{\text{arg}}, T_{\text{res}})$ is

$$\lambda u{:}\big[\text{cont:}(T_{\text{arg}} \to T_{\text{res}})\big]^\frown\big[\text{cont:}T_{\text{arg}}\big]\ .$$
$$\big[\text{cont=}u[0].\text{cont}@@u[1].\text{cont:}T_{\text{res}}\big]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \ \mid\ T'_{\text{daughter}_1}\big(@@T'_{\text{daughter}_2} : T_{\text{arg}}\big) : T_{\text{res}}$

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and T_{arg} and T_{res} are content types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \ \mid\ T'_{\text{daughter}_1}\big(@@T'_{\text{daughter}_2} : T_{\text{arg}}\big) : T_{\text{res}}$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \wedge\!\!\!\!\raisebox{-1ex}{:}\ \text{ContForwardApp}_{@@}(T_{\text{arg}}, T_{\text{res}})$$

$\alpha@_{\&}\beta$ **New!**

If T is a type, $\alpha : {}^{T}\!\textit{PPpty}$ and $\beta : {}^{T}\!\textit{PPpty}$ then *the property conjunction combination of* α *and* β, $\alpha@_{\&}\beta$, is

$$\lambda c{:}[\alpha.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{f}} \wedge\!\!\!\!\raisebox{-1ex}{:}\ \text{incr}([\beta.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{a}}, \alpha.\text{bg})\ .\ \alpha_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{f}}(c)\&\text{incr}([\beta]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{a}}, \alpha.\text{bg})(c)$$

ContForwardApp$_{@_{\&}}(T)$ **New!**

If T is a type, then ContForwardApp$_{@_{\&}}(T)$ is

$$\lambda u{:}\left[\ \text{cont} \ :\ {}^{T}\!\textit{PPpty}\ \right]^\frown\left[\ \text{cont} \ :\ {}^{T}\!\textit{PPpty}\ \right]\ .$$
$$\left[\ \text{cont}{=}u[0].\text{cont}@_{\&}u[1].\text{cont} \ :\ {}^{T}\!\textit{PPpty}\ \right]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ T'_{\text{daughter}_1}(@_{\&}T'_{\text{daughter}_2} : {}^{T}\!\textit{PPpty}) : {}^{T}\!\textit{PPpty}$ **New!**

If T is a type, T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types, $T_{\text{daughter}_1} \sqsubseteq \left[\ \text{cont} \ :\ {}^{T}\!\textit{PPpty}\ \right]$ and $T_{\text{daughter}_2} \sqsubseteq \left[\ \text{cont} \ :\ {}^{T}\!\textit{PPpty}\ \right]$, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ T'_{\text{daughter}_1}(@_{\&}T'_{\text{daughter}_2} : {}^{T}\!\textit{PPpty}) : {}^{T}\!\textit{PPpty}$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \wedge\!\!\!\!\raisebox{-1ex}{:}\ \text{ContForwardApp}_{@_{\&}}(T)$$

7.6.1.7 Action rules

(as in Chapter 6)

7.6.2 Universal speech act resources

(as in Chapter 2)

7.6.3 Universal dialogue resources

(as in Chapter 4)

7.6.4 English resources

7.6.4.1 Types and predicates

(as in Chapter 6)

7.6.4.2 Grammar

Lexical sign types

Let *Lexicon* be the set of lexical sign types defined inductively as follows. The following set is included in *Lexicon*.

$\{\mathrm{Lex}_{\mathrm{PropName}}$(“Dudamel”),
$\mathrm{Lex}_{\mathrm{PropName}}$(“Beethoven”),
$\mathrm{Lex}_{\mathrm{Pron}}$(“he”),
$\mathrm{Lex}_{\mathrm{numeral}}$(“nine”, 9),
$\mathrm{Lex}_{\mathrm{numeral}}$(“ninety”, 90),
$\mathrm{Lex}_{\mathrm{IndefArt}}$(“a”),
$\mathrm{Lex}_{\mathrm{Universal}}$(“every”),
$\mathrm{Lex}_{\mathrm{DefArt}}$(“the”),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$(“composer”, *Rec*, composer),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$(“conductor”, *Rec*, conductor),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$(“dog”, *Rec*, dog) (= $\Sigma_{\text{“dog”}}$),
RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“dog”}}$), *DogFrame*),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$(“passenger”, *Rec*, passenger) (= $\Sigma_{\text{“passenger”}}$),
RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“passenger”}}$), *PassengerFrame*),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$(“temperature”, *Rec*, temperature) (= $\Sigma_{\text{“temperature”}}$),
RestrictCommonNoun($\Sigma_{\text{“temperature”}}$, *AmbTempFrame*),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$(“leave”, *Rec*, leave),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$(“run”, *Rec*, run),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$(“rise”, $\left[\mathrm{c}{:}\mathit{TempRiseEventCntxt}\right]$, rise),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$(“rise”, $\left[\mathrm{c}{:}\mathit{PriceRiseEventCntxt}\right]$, rise),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$(“rise”, $\left[\mathrm{c}{:}\mathit{LocRiseEventCntxt}\right]$, rise),
$\mathrm{Lex}_{\mathrm{TransVerb}}$(“hug”, *Rec*, hug),
$\mathrm{Lex}_{\mathrm{TransVerb}}$(“find”, *Rec*, find),
$\mathrm{Lex}_{\mathrm{TransVerb}}$(“seek”, *Rec*, seek),
$\mathrm{Lex}_{\mathrm{TransVerb}}$(“worship”, *Rec*, worship),
$\mathrm{Lex}_{\mathrm{be_{ID}}}$(“is”),
$\mathrm{Lex}_{\mathrm{be_{scalar}}}$(“is”),
Lex(“ok”, *S*),
Lex(“aha”, *S*) $\}$

Transitive verbs as verb phrases New!

If Σ = $\mathrm{Lex}_{\mathrm{TransVerb}}(T_{\mathrm{Phon}}, T_{\mathrm{bg}}, p)$, for some T_{Phon}, T_{bg} and p, and $\Sigma \in$ *Lexicon*, then TransVerbToVerbPhrase(Σ) $\in$ *Lexicon*.

Constituent structure rule components

CnstrIsA

$$\lambda u{:}V\mathbin{\dot\wedge}\bigl[\text{s-event:}\bigl[\text{e:“is”}\bigr]\bigr]^\frown NP\mathbin{\dot\wedge}\left[\text{syn:}\begin{bmatrix}\text{daughters:}\mathit{Det}\mathbin{\dot\wedge}\bigl[\text{s-event:}\bigl[\text{e:“a”}\bigr]\bigr] \\ \quad {}^\frown N\mathbin{\dot\wedge}\bigl[\text{cont:}\mathit{Ppty}\bigr]\end{bmatrix}\right].$$
$$VP\mathbin{\dot\wedge}\bigl[\text{cont=}u\text{[2].syn.daughters[2].cont:}\mathit{Ppty}\bigr]$$

Constituent structure rules

Let *CSRules* be the set of constituent structure rules, defined inductively as follows. The following set is included in *CSRules*.

$$\{S \longrightarrow NP\ VP \mid NP'(VP'{:}Ppty){:}RecType,$$
$$NP \longrightarrow Det\ N \mid Det'({}_{@@}N'{:}PPpty){:}PQuant,$$
$$VP \longrightarrow V\ NP \mathbin{\dot\wedge} \text{CnstrIsA},$$
$$VP \longrightarrow V\ NP \mid V'(NP'{:}Quant){:}Ppty\}$$

Relative clauses New!

If i and j are natural numbers, then

$$Rel \longrightarrow NP_{\mathrm{wh}_i}\ S/j \mid NP'_{\mathrm{wh}_i}({}_{@_{\mathrm{wh}_{i,j}}} S/j'{:}PRecType){:}PPpty$$

is a member of *CSRules*

If T is a type, then

$$N \longrightarrow {}^{T}N\,{}^{T}Rel \;\mid\; {}^{T}N'({}_{@_{\&}}{}^{T}Rel' : {}^{T}PPpty) : {}^{T}PPpty$$

is a member of *CSRules*

7.7 Summary

In this chapter we looked at a witness-based account of quantification where we started by characterizing types of witness sets for quantifiers. This provides us with an account of generalized quantifiers which is more in line with the ideas expressed by Purver and Ginzburg (2004); Ginzburg and Purver (2008); Lücking and Ginzburg (2019) who argue that clarification and anaphoric phenomena in dialogue motivate an approach where quantifiers represent sets rather than sets of properties as in Montague's classical approach. We have argued that our approach suggests the possibility of accounting for a range of anaphoric phenomena and will provide some more detailed techniques for the treatment of anaphora in Chapter 8.

In addition our witness-based account provides a natural approach to a probabilistic account of quantification and we have suggested that estimations of probability are used in dialogue exchanges involving quantification in cases where it is obvious that the dialogue participants could not know whether the quantified statement is true but is able to estimate the probability of its truth based on the dialogue participant's previous experience or knowledge.

We presented an account of witness conditions for quantification ptypes based on two general patterns, one for monotone increasing quantifiers and one for monotone decreasing quantifiers. For a number of quantifiers there is in addition a specific witness condition, equivalent to the general one, which increases anaphoric possibilities and which also provide types more closely related to the DRT treatment of quantifiers.

Finally, in preparation for our treatment of quantifier scope ambiguity and binding in Chapter 8, we sketched a treatment of long distance dependencies where *wh*-phrases are treated as quantifiers which bind pronoun interpretations as arguments to verbs which are in the content but not realized syntactically. This is similar to proposals in categorial grammar (Steedman, 2012).

Chapter 8

Type-based underspecification

8.1 Introduction

In Section 8.2, we will explore a way of treating underspecification in terms of types of content rather than the standard view of underspecification in terms of underspecified representations corresponding to sets of contents.

In Section 8.3, we will extend this treatment of underspecification to include anaphoric readings.

8.2 Quantifier scope and underspecification

Given the kind of interpretation rules we have so far we can obtain a reading for (1a) which corresponds to the content in (1b), using the abbreviations 'boy′' as introduced in Chapter 5, example (61), **hug** for the parametric content of *hug* and **every**$^\frown$**dog** for the parametric content of *every dog*.

(1) a. a boy hugged every dog

b. $\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{c}{:}\left[\begin{array}{l}\text{f:}\left[\begin{array}{l}\text{f:}\textit{PropCntxt}\\ \text{a:}\textit{PropCntxt}\end{array}\right]\\ \text{a:}\left[\begin{array}{l}\text{f:}\textit{PropCntxt}\\ \text{a:}\left[\begin{array}{l}\text{f:}\textit{PropCntxt}\\ \text{a:}\textit{PropCntxt}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right] . \left[\begin{array}{lll}\text{restr=boy}' & : & \textit{Ppty}\\ \text{scope=}\textbf{hug}(c)(\textbf{every}{}^\frown\textbf{dog}(c))|_{\mathcal{F}(\text{boy}')} & : & \textit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{array}\right]\urcorner$

This, of course, represents the reading where there is a boy such that he hugs every dog. In order to obtain the reading where *every dog* has wide scope, we follow Montague and pretty much everybody else in basing our treatment of quantifier scope on the treatment of free pronouns, though without the contribution of any gender information. Let us imagine just for a moment that such a pronoun existed in English and is written as *it** with the kind of pronoun interpretations given in Chapter 4, Section 4.6. Then the content for (2a) could be (2b).

(2) a. a boy hugged it*

b. $\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{s}{:}\left[\text{x}_0{:}\textit{Ind}\right]\\ \mathfrak{c}{:}\left[\begin{array}{l}\text{f:}\left[\begin{array}{l}\text{f:}\textit{PropCntxt}\\ \text{a:}\textit{PropCntxt}\end{array}\right]\\ \text{a:}\left[\begin{array}{l}\text{f:}\textit{PropCntxt}\\ \text{a:}\textit{PropCntxt}\end{array}\right]\end{array}\right]\end{array}\right] . \left[\begin{array}{lll}\text{restr=boy}' & : & \textit{Ppty}\\ \text{scope=}\textbf{hug}(c)(\lambda P{:}\textit{Ppty}\ .\ P\{c.\mathfrak{s}.\text{x}_0\})|_{\mathcal{F}(\text{boy}')} & : & \textit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{array}\right]$

We will refer to (2b) as ‘$\textbf{a}{}^\frown\textbf{boy}{}^\frown\textbf{hugged}{}^\frown\textbf{it}^*$’. Let us further imagine, contrary to fact, that English represented the fact that a noun phrase has wide scope over a sentence by placing it at the beginning of a sentence as in (3a) and giving it an interpretation where the interpretation of *it** gets bound as in (3b).

(3) a. every dog, a boy hugged it*

$$\text{b. } \lambda c{:}\begin{bmatrix} \mathit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix} \text{f:}\begin{bmatrix}\text{f:}\mathit{PropCntxt} \\ \text{a:}\mathit{PropCntxt}\end{bmatrix} \\ \text{a:}\begin{bmatrix} \text{f:}\begin{bmatrix}\text{f:}\mathit{PropCntxt} \\ \text{a:}\mathit{PropCntxt}\end{bmatrix} \\ \text{a:}\begin{bmatrix}\text{f:}\mathit{PropCntxt} \\ \text{a:}\mathit{PropCntxt}\end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} .$$

$$\begin{bmatrix} \text{restr=dog}' & : & \mathit{Ppty} \\ \text{scope=}\lambda r{:}\begin{bmatrix}\text{x:}\mathit{Ind}\end{bmatrix} \ . \ \textbf{a}^\frown\textbf{boy}^\frown\textbf{hugged}^\frown\textbf{it}^* \ (c[\mathfrak{s}.\text{x}_0 = r.\text{x}])|_{\mathcal{F}(\text{dog}')} & : & \mathit{Ppty} \\ \text{e} & : & \text{every(restr, scope)} \end{bmatrix}$$

The imaginary English expression (3a) corresponds quite closely to the kind of representation for wide scope readings that are used in various theories of logical form. A major difference is that in logical form there is an index corresponding to the label 'x_0' that we use in the interpretation which shows that *every dog* binds *it**. This might be represented something like in (4).

(4) every dog$_{x_0}$, a boy hugged it$^*_{x_0}$

The imaginary sentence (3a) also corresponds closely to Montague's (1973) treatment of scope phenomena. Montague would also index the pronoun and use a quantification rule with the same index which would replace the pronoun with the noun-phrase being quantified in.

Neither of these options are open to us since our syntax is defined in terms of types of utterance situations and signs which relate utterance situations to contents. Our realistic strategy does not allow for the use of additional imaginary utterance structures. For this reason we will adapt the kind of storage technique used in Cooper (1983). In the original version of storage we moved from assigning a single content to a syntactic structure to assigning a set of contents in order to allow for the ambiguous interpretation of a single syntactic structure. In our sign-based approach using types the corresponding move is not such a major change and the result yields a theory involving underspecified content rather than a set of contents.

To see this consider the type *Sign* introduced in Chapter 3. Any object of type *Sign* will be of the type in (5).

$$(5) \quad \begin{bmatrix} \text{s-event} & : & \mathit{SEvent} \\ \text{syn} & : & \mathit{Syn} \\ \text{cont} & : & \mathit{Cont} \end{bmatrix}$$

The type in (5) is completely underspecified. Any sign will be of this type. We could specify it with respect to content by making the 'cont'-field be a manifest field as in (6), where φ is a particular content.

$$(6) \quad \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{syn} & : & \textit{Syn} \\ \text{cont}{=}\varphi & : & \textit{Cont} \end{bmatrix}$$

Now recall that the manifest field $\left[\text{cont}{=}\varphi{:}\textit{Cont}\right]$ is just a convenient way of writing $\left[\text{cont}{:}\textit{Cont}_\varphi\right]$ where $\textit{Cont}_\varphi$ is a singleton type whose only witness is φ if $\varphi : \textit{Cont}$; otherwise it has no witnesses. It is in this sense that the content has been specified to be φ. Suppose now that we do not have enough information to fully specify the content, that is, tie it down to be one particular content, but we know that it has to be one of either φ or ψ. This could be represented by using a join type of two singleton types, $\textit{Cont}_\varphi \vee \textit{Cont}_\psi$, as in (7).

$$(7) \quad \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{syn} & : & \textit{Syn} \\ \text{cont} & : & \textit{Cont}_\varphi \vee \textit{Cont}_\psi \end{bmatrix}$$

(7) is the type of signs whose contents are either φ or ψ. This is, then, a single type, which corresponds to an "underspecified content". Of course, the set of witnesses of the join type, $\{\varphi, \psi\}$, could correspond to the set of contents that could be generated by a storage algorithm. Another way to achieve this is to use the operator, $\mathfrak{T}$, introduced in Chapter 7, example (18) on p. 356 which takes a set and returns a type whose witnesses are exactly the members of the set. We could thus use (8) instead of (7) with the same effect.

$$(8) \quad \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{syn} & : & \textit{Syn} \\ \text{cont} & : & \mathfrak{T}(\{\varphi, \psi\}) \end{bmatrix}$$

Our strategy is to devise a way of computing such types on the basis of compositional interpretation without having to enumerate the members of the set of contents in the way that we did in (7) and (8).

We will exploit our treatment of context to include a store of parametric quantifiers that are to be given a wide scope interpretation. Thus our characterization of the type *Cntxt* will be extended to (9).

(9) $\begin{bmatrix} \mathfrak{q} & : & \textit{QStore} \\ \mathfrak{s} & : & \textit{Assgnmnt} \\ \mathfrak{w} & : & \textit{Assgnmnt} \\ \mathfrak{g} & : & \textit{Assgnmnt} \\ \mathfrak{c} & : & \textit{PropCntxt} \end{bmatrix}$

QStore is the type of assignments all of whose values are parametric quantifiers, that is, as characterized in (10).

(10) *QStore* is a basic type. r : *QStore* iff $r : Assgnmnt$ and for any $\mathrm{x}_i \in \mathrm{labels}(r)$, $r.\mathrm{x}_i$: *PQuant*

An example of a witness for *QStore* would thus be (11) where we have stored the basic parametric content of *every dog* under the label 'x_0'.

(11) $\left[\ \mathrm{x}_0 \ = \ \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix} . \ \lambda P{:}\textit{Ppty} \ . \ \begin{bmatrix}\mathrm{restr{=}dog'}{:}\textit{Ppty}\\ \mathrm{scope}{=}P{:}\textit{Ppty}\\ \mathrm{e{:}every(restr,\ scope)}\end{bmatrix}\urcorner\ \right]$

The record in (11) is, among other types, of the type in (12), which will be the kind of type we will be using in our content types which will be underspecified for quantifier scope.

(12) $\begin{bmatrix}\textit{QStore}\\ \mathrm{x}_0{=}\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix} . \ \lambda P{:}\textit{Ppty} \ . \ \begin{bmatrix}\mathrm{restr{=}dog'}{:}\textit{Ppty}\\ \mathrm{scope}{=}P{:}\textit{Ppty}\\ \mathrm{e{:}every(restr,\ scope)}\end{bmatrix}\urcorner \ : \ \textit{PQuant}\end{bmatrix}$

Suppose that the basic parametric content associated with a noun-phrase is $\mathcal{Q}$, that is, the type of the content as we have been expressing it so far (using a manifest field) will be $\textit{Cont}_{\mathcal{Q}}$. Now we want to generalize this type so that not only $\mathcal{Q}$ will be a witness for the type but also a parametric content where $\mathcal{Q}$ is required in the qstore of the context. This parametric quantifier will be (13).

(13) $\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{q}{:}\begin{bmatrix}\mathrm{x}_0{=}\mathcal{Q}{:}\textit{PQuant}\end{bmatrix}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\end{bmatrix}\end{bmatrix} . \ \lambda P{:}\textit{Ppty} \ . \ P\{c.\mathfrak{s}.\mathrm{x}_0\}\urcorner$

Note that (13) is exactly like the pronoun content introduced in Chapter 4, example (96), except that we have a parametric quantifier required in the qstore labelled with the same label 'x_0' as is used in the pronoun content. Thus this corresponds to the content of *it** in (2) and (3). Note that (13) is of type *PQuant*. It is distinguished from a parametric quantifier like (14), however, in that it requires the qstore in the context to be non-empty.

(14) $\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{f:}\textit{PropCntxt}\\ \text{a:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ \begin{bmatrix}\text{restr=dog}'{:}\textit{Ppty}\\ \text{scope=}P{:}\textit{Ppty}\\ \text{e:every(restr, scope)}\end{bmatrix}\urcorner$

Using terminology that goes back to Bos (1996), we will say that (14) is *plugged* (that is, in our terms, does not require anything to be in the qstore) whereas (13) is *unplugged* (that is, in our terms, *does* require something to be in the qstore). The intuition is that when a quantifier has been placed in the qstore it has been unplugged from the main interpretation and needs to be plugged back in at some point in order to get a fully specified interpretation. We formally characterize the notion *plugged* in (15).

(15) A parametric content, α, is *unplugged* iff $c : \alpha$.bg implies $c.\mathfrak{q} \neq \emptyset$ (that is, $c.\mathfrak{q}$ is not the empty record). Otherwise α is *plugged*.

We can derive (13) from $\mathcal{Q}$ by an operation 'store' characterized in (16).

(16) If $\mathcal{Q}$: *PQuant* and $\mathcal{Q}$ is plugged, then $\mathrm{store}(\mathcal{Q})$ is

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{q}{:}\begin{bmatrix}\text{x}_0\text{=}\mathcal{Q}{:}\textit{PQuant}\end{bmatrix}\\ \mathfrak{s}{:}\begin{bmatrix}\text{x}_0{:}\textit{Ind}\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P\{c.\mathfrak{s}.\text{x}_0\}\urcorner$$

We will now turn our attention to retrieval which removes a quantifier from the qstore and quantifies over the virtual pronoun created by storage.

In order to do this we need to generalize our characterization of '$\ominus$' in Chapter 7, example (144) to include *QStore* as a special case exactly similar to *Assgnmnt*. This is done in (17).

(17)
1. If T is *Assgnmnt*$\wedge T'$, T' is a record type and $\ell \in \mathrm{labels}(T')$, then
   a) if $\mathrm{labels}(T') = \{\ell\}$, $T \ominus \ell =$ *Assgnmnt*
   b) otherwise, $T \ominus \ell =$ *Assgnmnt* $\wedge\ (T' \ominus \ell/T)$
2. If T is *PropCntxt*$\wedge T'$ and $\pi \in \mathrm{tpaths}(T')$ then
   a) if π is ℓ and $\mathrm{labels}(T') = \{\ell\}$, then $T \ominus \pi =$ *PropCntxt*

b) otherwise, $T \ominus \pi = \textit{PropCntxt} \wedge (T' \ominus \pi/T')$

3. If T is $\textit{QStore}{\wedge}T'$ and $\pi \in \mathrm{tpaths}(T')$ then

   a) if π is ℓ and $\mathrm{labels}(T') = \{\ell\}$, then $T \ominus \pi = \textit{PropCntxt}$

   b) otherwise, $T \ominus \pi = \textit{QStore} \wedge (T' \ominus \pi/T')$

We now characterize retrieval in a version corresponding to quantification with scope over sentences. In a more complete treatment we would at least add quantification with scope over verb phrases and common nouns corresponding to Montague's (1973) treatment. In (18), we characterize an operation 'retrieve' which maps a parametric record type with a quantifier in store to one where the quantifier is removed from the store and given scope over the non-parametric content.

(18) If α : *PRecType*, $\mathcal{Q}$: *PQuant* $\alpha.\mathrm{bg} \sqsubseteq \left[\mathfrak{q}{:}\left[\mathrm{x}_i{=}\mathcal{Q}{:}\textit{Ind}\right]\right]$, $\mathcal{Q}'$ is $[\mathcal{Q}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{f}}$ and α' is $[\mathrm{incr}(\alpha, \mathcal{Q}')]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.a}$, then $\mathrm{retrieve}(\mathrm{x}_i, \alpha)$ is

$$\lambda c{:}(\mathcal{Q}'.\mathrm{bg}\underset{\cdot}{\wedge}\alpha'.\mathrm{bg} \ominus \mathfrak{q}.\mathrm{x}_i, \mathfrak{s}.\mathrm{x}_i) \;.$$
$$\mathcal{Q}'(c)(\mathfrak{P}(\ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\textit{Ind}\\ \mathfrak{s}{:}\begin{bmatrix}\textit{Assgnmnt}\\ \mathrm{x}_i{=}\Uparrow\mathrm{x}{:}\textit{Ind}\end{bmatrix}\end{bmatrix}\underset{\cdot}{\wedge}\alpha'.\mathrm{bg}^{\mathfrak{s}.\mathrm{x}_i} \;.\; \alpha'(c[r][\mathfrak{q}.\mathrm{x}_i = \mathcal{Q}])\urcorner))$$

'retrieve' applies to a label, 'x_i' and a parametric record type, α, which contains a parametric quantifier, $\mathcal{Q}$, in its qstore labelled by 'x_i'. It returns a parametric record type where the context type is an appropriate combination of the context types (labelled 'bg') associated with $\mathcal{Q}$ and α including path adjustment and incrementation but with fields labelled '$\mathfrak{q}.\mathrm{x}_i$' and '$\mathfrak{s}.\mathrm{x}_i$' removed together with any fields depending on them. The field labelled '$\mathfrak{s}.\mathrm{x}_i$' and fields dependent on it are added to the domain type of the property to which $\mathcal{Q}$ is applied and a '$\mathfrak{q}.\mathrm{x}_i$'-field is added to the context argument for α' to make the context an appropriate argument, though this field will not appear in the result. Purification ($\mathfrak{P}$) is applied to the property so that the additional fields from the domain type are "moved down" into α giving the effect of what in Discourse Representation Theory would be called local accommodation of any presuppositions (Van der Sandt, 1992). This means that any presuppositions associated with the quantifier will apply at the level at which it is quantified in.

Now we have two ways, storage and retrieval, in which we can derive parametric contents from other parametric contents. How can we, then, characterize the type of parametric contents associated with some particular phrase? We will introduce a type *ContType* characterized in (19).

(19) a. *ContType*, "the type of types of contents", is a basic type

b. T : *ContType* iff $T \sqsubseteq$ *Cont*

Suppose that T : *ContType*. Then we define a new type $\mathfrak{S}(T)$ whose witnesses are the closure of the set of witnesses of T under 'store' and '$\mathrm{retrieve}$'. We give a precise characterization of this in (20).

(20) a. If T : *ContType*, then $\mathfrak{S}(T)$ is a type

b. The witnesses of $\mathfrak{S}(T)$ are characterized by

1. if $\varphi : T$ then $\varphi : \mathfrak{S}(T)$
2. if $\varphi : \mathfrak{S}(T)$ and φ is in the range of 'store', then $\mathrm{store}(\varphi) : \mathfrak{S}(T)$
3. if $\varphi : \mathfrak{S}(T)$ and 'x_i' and φ are appropriate arguments to '$\mathrm{retrieve}$', then $\mathrm{retrieve}(\mathrm{x}_i, \varphi) : \mathfrak{S}(T)$
4. nothing is a witness for $\mathfrak{S}(T)$ except as required above.

If φ is a parametric content, that is, φ : *Cont*, we use the notation $\varphi^{\mathfrak{S}}$ to represent $\mathfrak{S}(\mathit{Cont}_{\varphi})$, that is, the type whose witnesses are the closure of $\{\varphi\}$ under 'store' and '$\mathrm{retrieve}$'.

How should such types of parametric contents be combined in compositional semantics? First, the value in the 'cont'-field in a sign will now not be a parametric content as previously but a type of parametric contents, that is, it will be of type *ContType*. Thus we redefine the type *Sign* as in (21).

(21) a. *Sign*, "the type of signs", is a basic type

b. σ : *Sign* iff $\sigma : \begin{bmatrix} \text{s-event} & : & \mathit{SEvent} \\ \text{syn} & : & \mathit{Syn} \\ \text{cont} & : & \mathit{ContType} \end{bmatrix}$

Suppose that T_1 and T_2 are of type *ContType* and that $\mathcal{O}$ is a combination operation such as @, @@, $@_{\mathrm{wh}_{i,j}}$, then we say $T_1\mathcal{O}^{\mathfrak{S}}T_2$ is also a type with the witness condition in (22).

(22) If $\alpha : T_1$, $\beta : T_2$ and $\alpha\mathcal{O}\beta$ is defined, then $\alpha\mathcal{O}\beta : T_1\mathcal{O}^{\mathfrak{S}}T_2$. Nothing else is a witness for $T_1\mathcal{O}^{\mathfrak{S}}T_2$.

What we need then for the type of parametric contents for the combined constituents is $\mathfrak{S}(T_1\mathcal{O}^{\mathfrak{S}}T_2)$, that is, the type of the closure of the set of all combinations under 'store' and '$\mathrm{retrieve}$'. Note that (22) has the consequence that if '$\alpha\mathcal{O}\beta$' is undefined for all witnesses, α and β of T_1 and T_2 respectively we will still obtain a result for $T_1\mathcal{O}^{\mathfrak{S}}T_2$, albeit a type which has no witnesses.

This will mean that we do not have to be as careful in keeping track of typing when interpreting syntactic constructions, though with the consequence that some phrases will not have any content.

In (23) we introduce versions of 'ContForwardApp' operations which apply at the content type level.

(23) If $\mathcal{O}$ is one of @, @@, $@_{\text{wh}_{i,j}}$ (for some natural numbers i and j) or $@_{\&}$, then $\text{ContForwardApp}_{\mathfrak{S},\mathcal{O}}$ is

$$\lambda u{:}\big[\text{cont:}\textit{ContType}\big]^\frown\big[\text{cont:}\textit{ContType}\big]\ .\ \big[\text{cont:}\mathfrak{S}(u[0].\text{cont}\mathcal{O}^{\mathfrak{S}}u[1].\text{cont})\big]$$

Note that we now no longer have to keep track of the arguments to ContForwardApp that we had in some of the variants since the combination of the types will always return a result, though if the types of contents do not match for the particular combination operation the combined type will have no witnesses. This means that we can simplify our notation for constituent structure rules as in (24).

(24) If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and $\mathcal{O}$ is a combination operation, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ T'_{\text{daughter}_1}(_{\mathcal{O}}T'_{\text{daughter}_2})$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \dot{\wedge}\ \text{ContForwardApp}_{\mathfrak{S},\mathcal{O}}$$

8.3 Anaphora

We will treat anaphora by adding to the storage mechanism we have just introduced. In informal terms the idea is that if the content type of an utterance has something corresponding to (25a) as a witness then it will also have something corresponding to (25b) as a witness where x_1 has been anaphorically related to x_0.

(25) a. x_0 thinks that x_1 has succeeded

b. x_0 thinks that x_0 has succeeded

Thus in general the content type yielded by the grammatical resources will be underspecified as to whether there is anaphora or not but will nevertheless delimit what the anaphoric possibilities are. Anaphora will be accounted for at the point of combination. This is illustrated schematically in (26).

(26) a. x_o + thinks that x_1 has succeeded = x_0 thinks that x_1 has succeeded

b. If 'x_o + thinks that x_1 has succeeded' is an interpretation, then 'x_o + (thinks that x_1 has succeeded)[$x_1 \rightsquigarrow x_0$]' is an interpretation

We will, of course, not be implementing this in terms of replacing variables as in (26) but rather in adjusting the contexts of interpretation associated with pronoun utterances. For example, we will define a variant of the combination operation '@', '$@_{i,j}$' which anaphorically relates a pronoun associated with the context path '$\mathfrak{s}.j$' to one associated with the context path '$\mathfrak{s}.i$'. This is given in (27).

(27) If $\mathcal{O}$ is a combination operator, then so is $\mathcal{O}_{i,j}$, where i and j are natural numbers.

If α and β are parametric contents such that $\alpha\mathcal{O}\beta$ is defined, α.bg $\sqsubseteq \left[\mathfrak{s}{:}\left[\mathrm{x}_i{:}\textit{Ind}\right]\right]$ and $\mathrm{incr}(\beta.\mathrm{bg}, \alpha.\mathrm{bg}) \sqsubseteq \left[\mathfrak{s}{:}\left[\mathrm{x}_j{:}\textit{Ind}\right]\right]$ but $\mathrm{incr}(\beta.\mathrm{bg}, \alpha.\mathrm{bg}) \not\sqsubseteq \left[\mathfrak{q}{:}\left[\mathrm{x}_j{:}\textit{PQuant}\right]\right]$, then $\alpha\mathcal{O}_{i,j}\beta$ is

$$[\alpha\mathcal{O}\beta]_{\mathfrak{s}.\mathrm{x}_j \rightsquigarrow \mathfrak{s}.\mathrm{x}_i}$$

We can now add contents with anaphora to our characterization of $\mathfrak{S}(T)$ given in (20) as in (28) where we again use boxing to indicate the new material.

(28) a. If T : *ContType*, then $\mathfrak{S}(T)$ is a type

b. The witnesses of $\mathfrak{S}(T)$ are characterized by

1. if $\varphi : T$ then $\varphi : \mathfrak{S}(T)$
2. $\boxed{\text{if } \alpha\mathcal{O}\beta : \mathfrak{S}(T), \text{ (for some combination operation, } \mathcal{O}\text{) and } \alpha\mathcal{O}_{i,j}\beta \text{ is defined (for some natural numbers, } i \text{ and } j\text{), then } \alpha\mathcal{O}_{i,j}\beta : \mathfrak{S}(T)}$
3. if $\varphi : \mathfrak{S}(T)$ and φ is in the range of 'storage', then $\mathrm{storage}(\varphi) : \mathfrak{S}(T)$
4. if $\varphi : \mathfrak{S}(T)$ and 'x_i' and φ are appropriate arguments to 'retrieve', then $\mathrm{retrieve}(\mathrm{x}_i, \varphi) : \mathfrak{S}(T)$
5. nothing is a witness for $\mathfrak{S}(T)$ except as required above.

We will now take some examples of key anaphoric phenomena and discuss how we could use these tools to account for them.

No girl thinks she failed Given the strategy we suggested in Section 4.6 for interpreting unbound pronouns the foreground of a content for *she failed* would be parallel to example (96c) as in (29), where we in addition express this as the content type obtained by $\mathfrak{S}$.

(29) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{s}{:}\left[\mathrm{x}_0{:}\textit{Ind}\right]\\ \mathfrak{c}{:}\left[\begin{array}{l}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{array}\right]\end{array}\right] . \left[\ \mathrm{e}\ :\ \mathrm{fail}(c.\mathfrak{s}.\mathrm{x}_0)\ \right]\urcorner^{\mathfrak{S}}$$

Call this **she**$^\frown$**failed**. Then the content type for *thinks she failed* is (30).

(30) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{s}{:}\left[\mathrm{x}_0{:}\textit{Ind}\right]\\ \mathfrak{c}{:}\left[\begin{array}{l}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\left[\begin{array}{l}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{array}\right]\end{array}\right]\end{array}\right] . \ulcorner\lambda r{:}\left[\mathrm{x}{:}\textit{Ind}\right] . \left[\ \mathrm{e}\ :\ \mathrm{think}(r.\mathrm{x},\ \left[\mathrm{e}{:}\mathrm{fail}(c.\mathfrak{s}.\mathrm{x}_0)\right])\ \right]\urcorner\urcorner^{\mathfrak{S}}$$

Call this **thinks**$^\frown$**she**$^\frown$**failed**. Here *she* is still a free occurrence of a pronoun dependent on the context for resolution.

The content type associated with *no girl* will be (31).

(31) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{c}{:}\left[\begin{array}{l}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{array}\right]\end{array}\right] . \lambda P{:}\textit{Ppty} . \left[\begin{array}{l}\mathrm{restr{=}girl'}{:}\textit{Ppty}\\ \mathrm{scope}{=}P{:}\textit{Ppty}\\ \mathrm{e{:}no(restr,\ scope)}\end{array}\right]\urcorner^{\mathfrak{S}}$$

Let us represent the *generator* of this type, that is, (32), by ‘no′(girl′)’.

(32) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{c}{:}\left[\begin{array}{l}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{array}\right]\end{array}\right] . \lambda P{:}\textit{Ppty} . \left[\begin{array}{l}\mathrm{restr{=}girl'}{:}\textit{Ppty}\\ \mathrm{scope}{=}P{:}\textit{Ppty}\\ \mathrm{e{:}no(restr,\ scope)}\end{array}\right]\urcorner$$

This means that one witness for the type (31) is (33) where ‘no′(girl′)’ has been stored.

(33) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{q}{:}\left[\mathrm{x}_0{=}\mathrm{no'(girl')}{:}\textit{PQuant}\right]\\ \mathfrak{s}{:}\left[\mathrm{x}_0{:}\textit{Ind}\right]\end{array}\right] . \lambda P{:}\textit{Ppty} . P\{c.\mathfrak{s}.\mathrm{x}_0\}\urcorner$$

Note that the context type for this parametric content has the path ‘$\mathfrak{s}.\mathrm{x}_0$’ which means that it is available for anaphoric version of combination operations, thus enabling *she* in *thinks she failed* to be related to ‘$\mathfrak{s}.\mathrm{x}_0$’. This can be achieved by the combination of parametric contents expressed in (34a) which is identical with (34b). This uses ‘$@_{0,1}$’ as characterized in (27).

$$\text{(34) a. } \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{q}{:}\begin{bmatrix}\mathrm{x}_0{=}\mathrm{no}'(\mathrm{girl}'){:}\textit{PQuant}\end{bmatrix}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\end{bmatrix}\end{bmatrix} . \lambda P{:}\textit{Ppty} . P\{c.\mathfrak{s}.\mathrm{x}_0\}\urcorner$$

$$@_{0,1}$$

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\end{bmatrix}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix} . \ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\textit{Ind}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & \mathrm{think}(r.\mathrm{x}, \begin{bmatrix}\mathrm{e}{:}\mathrm{fail}(c.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix})\end{bmatrix}\urcorner\urcorner$$

$$\text{b. } \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{q}{:}\begin{bmatrix}\mathrm{x}_0{=}\mathrm{no}'(\mathrm{girl}'){:}\textit{PQuant}\end{bmatrix}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\end{bmatrix}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & \mathrm{think}(c.\mathfrak{s}.\mathrm{x}_0, \begin{bmatrix}\mathrm{e}{:}\mathrm{fail}(c.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix})\end{bmatrix}\urcorner$$

Application of 'retrieve' to the content (34) will obtain a content where the scope of the quantifier is the property of "being a girl who thinks she (the girl) failed". The whole content is given in (35) where (35a–c) are identical.

$$\text{(35) a. } \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix} .$$

$$(\lambda P{:}\textit{Ppty} . \begin{bmatrix}\mathrm{restr}{=}\mathrm{girl}' & : & \textit{Ppty}\\ \mathrm{scope}{=}P|_{\mathcal{F}(\mathrm{girl}')} & : & \textit{Ppty}\\ \mathrm{e} & : & \mathrm{no}(\mathrm{restr}, \mathrm{scope})\end{bmatrix}$$

$$(\mathfrak{P}(\ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\textit{Ind}\\ \mathfrak{s}{:}\begin{bmatrix}\textit{Assgnmnt}\\ \mathrm{x}_0{=}\Uparrow\mathrm{x}{:}\textit{Ind}\end{bmatrix}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & \mathrm{think}(r.\mathfrak{s}.\mathrm{x}_0, \begin{bmatrix}\mathrm{e}{:}\mathrm{fail}(r.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix})\end{bmatrix}\urcorner)))\urcorner$$

$$\text{b. } \ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix} .$$

$$(\lambda P{:}Ppty\ .\ \begin{bmatrix} \text{restr=girl}' & : & Ppty \\ \text{scope=}P|_{\mathcal{F}(\text{girl}')} & : & Ppty \\ \text{e} & : & \text{no(restr, scope)} \end{bmatrix}$$

$$(\ulcorner\lambda r{:}\begin{bmatrix}\text{x}{:}Ind\end{bmatrix}\ .\ \begin{bmatrix} \mathfrak{c}{:}\begin{bmatrix}\mathfrak{s}{:}\begin{bmatrix}Assgnmnt \\ \text{x}_0\text{=}r.\text{x}{:}Ind\end{bmatrix}\end{bmatrix} \\ \text{e:think}(\mathfrak{c}.\mathfrak{s}.\text{x}_0,\ \begin{bmatrix}\text{e:fail}(\mathfrak{c}.\mathfrak{s}.\text{x}_0)\end{bmatrix}) \end{bmatrix}\urcorner))\urcorner \qquad \text{(purification)}$$

c. $$\ulcorner\lambda c{:}\begin{bmatrix} Cntxt \\ \mathfrak{c}{:}\begin{bmatrix} \text{f:}PropCntxt \\ \text{a:}\begin{bmatrix} \text{f:}PropCntxt \\ \text{a:}\begin{bmatrix} \text{f:}PropCntxt \\ \text{a:}\begin{bmatrix}\text{f:}PropCntxt \\ \text{a:}PropCntxt\end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}\ .$$

$$\begin{bmatrix} \text{restr=girl}' & : & Ppty \\ \text{scope=}\ulcorner\lambda r{:}\begin{bmatrix}\text{x}{:}Ind \\ \text{e:girl(x)}\end{bmatrix}\ .\ \begin{bmatrix} \mathfrak{c}{:}\begin{bmatrix}\mathfrak{s}{:}\begin{bmatrix}Assgnmnt \\ \text{x}_0\text{=}r.\text{x}{:}Ind\end{bmatrix}\end{bmatrix} \\ \text{e:think}(\mathfrak{c}.\mathfrak{s}.\text{x}_0,\ \begin{bmatrix}\text{e:fail}(\mathfrak{c}.\mathfrak{s}.\text{x}_0)\end{bmatrix}) \end{bmatrix}\urcorner & : & Ppty \\ \text{e} & : & \text{no(restr, scope)} \end{bmatrix}\urcorner$$

(β-reduction, property restriction)

A man walked. He whistled. Given our strategy for defining the content of quantified sentences in terms of generalized quantifiers a witness for the content type of *a man walked* will be (36).

(36) $$\ulcorner\lambda c{:}\begin{bmatrix} Cntxt \\ \mathfrak{c}{:}\begin{bmatrix} \text{f:}\begin{bmatrix}\text{f:}PropCntxt \\ \text{a:}PropCntxt\end{bmatrix} \\ \text{a:}PropCntxt \end{bmatrix} \end{bmatrix}\ .\ \begin{bmatrix} \text{restr=man}' & : & Ppty \\ \text{scope=walk}'|_{\mathcal{F}(\text{restr})} & : & Ppty \\ \text{e} & : & \text{exist(restr, scope)} \end{bmatrix}\urcorner$$

We know from our treatment of the witness conditions associated with 'exist' that (36) is equivalent to (37).

(37) $$\ulcorner\lambda c{:}\begin{bmatrix} Cntxt \\ \mathfrak{c}{:}\begin{bmatrix} \text{f:}\begin{bmatrix}\text{f:}PropCntxt \\ \text{a:}PropCntxt\end{bmatrix} \\ \text{a:}PropCntxt \end{bmatrix} \end{bmatrix}\ .$$

$$\begin{bmatrix} \text{restr=man}' & : & Ppty \\ \text{scope=walk}'|_{\mathcal{F}(\text{restr})} & : & Ppty \\ \text{e} & : & \begin{bmatrix} \text{x} & : & \mathfrak{T}(\Uparrow\text{restr}) \\ \text{e} & : & \mathfrak{P}(\Uparrow\text{scope})\{\text{x}\} \end{bmatrix} \end{bmatrix}\urcorner$$

Using our previous treatment for free pronouns, *he whistled* will have (38) as a witness of its content type.

(38) $$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\mathit{PropCntxt}\\ \mathrm{a}{:}\mathit{PropCntxt}\end{bmatrix}\end{bmatrix}\ .\ \begin{bmatrix}\mathrm{e} & : & \mathrm{whistle}(c.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner$$

We will represent (38) as **he⌒whistled**.

The utterance of *He whistled* is to be interpreted in the context of the previous utterance of *a man walked*. We will achieve this by merging the quasi-fixed point type (see Chapter 5, p. 245) of the foreground of the parametric content of the previous utterance with the context type of the current utterance. The quasi-fixed point type for the foreground of (37) is (39).

(39) $$\begin{bmatrix}\mathfrak{c}^* & : & \begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\begin{bmatrix}\mathrm{f}{:}\mathit{PropCntxt}\\ \mathrm{a}{:}\mathit{PropCntxt}\end{bmatrix}\\ \mathrm{a}{:}\mathit{PropCntxt}\end{bmatrix}\end{bmatrix}\\ \mathrm{restr{=}man}' & : & \mathit{Ppty}\\ \mathrm{scope{=}walk}'|_{\mathcal{F}(\mathrm{restr})} & : & \mathit{Ppty}\\ \mathrm{e} & : & \begin{bmatrix}\mathrm{x} & : & \mathfrak{T}(\Uparrow\mathrm{restr})\\ \mathrm{e} & : & \mathfrak{P}(\Uparrow\mathrm{scope})\{\mathrm{x}\}\end{bmatrix}\end{bmatrix}$$

We will merge this under the label '𝔭' ("previous") into the context type in (38) yielding (40).

(40) $$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{p}{:}\begin{bmatrix}\mathfrak{c}^*{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\begin{bmatrix}\mathrm{f}{:}\mathit{PropCntxt}\\ \mathrm{a}{:}\mathit{PropCntxt}\end{bmatrix}\\ \mathrm{a}{:}\mathit{PropCntxt}\end{bmatrix}\end{bmatrix}\\ \mathrm{restr{=}man}'{:}\mathit{Ppty}\\ \mathrm{scope{=}walk}'|_{\mathcal{F}(\mathrm{restr})}{:}\mathit{Ppty}\\ \mathrm{e}{:}\begin{bmatrix}\mathrm{x}{:}\mathfrak{T}(\Uparrow\mathrm{restr})\\ \mathrm{e}{:}\mathfrak{P}(\Uparrow\mathrm{scope})\{\mathrm{x}\}\end{bmatrix}\end{bmatrix}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}\ .\ \begin{bmatrix}\mathrm{e} & : & \mathrm{whistle}(c.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner$$

(40) makes the content of the previous utterance be part of the context for content of the current utterance. It does not, however, express the anaphoric relation between *he* and *a man*. In order

to do this we need to require that $\mathfrak{s}.\mathrm{x}_0$ in the context is identical with $\mathfrak{p}$.e.s. This can be done by introducing a manifest field under the '$\mathfrak{s}$'-label as in (41).

(41)
$$\ulcorner\lambda c{:}\left[\begin{array}{l} \mathit{Cntxt} \\ \mathfrak{p}{:}\left[\begin{array}{l} \mathfrak{c}^*{:}\left[\begin{array}{l} \mathit{Cntxt} \\ \mathfrak{c}{:}\left[\begin{array}{l} \mathrm{f}{:}\left[\begin{array}{l} \mathrm{f}{:}\mathit{PropCntxt} \\ \mathrm{a}{:}\mathit{PropCntxt} \end{array}\right] \\ \mathrm{a}{:}\mathit{PropCntxt} \end{array}\right] \end{array}\right] \\ \mathrm{restr}{=}\mathrm{man}'{:}\mathit{Ppty} \\ \mathrm{scope}{=}\mathrm{walk}'|_{\mathcal{F}(\mathrm{restr})}{:}\mathit{Ppty} \\ \mathrm{e}{:}\left[\begin{array}{l} \mathrm{x}{:}\mathfrak{T}(\Uparrow\mathrm{restr}) \\ \mathrm{e}{:}\mathfrak{P}(\Uparrow\mathrm{scope})\{\mathrm{x}\} \end{array}\right] \end{array}\right] \\ \mathfrak{s}{:}\left[\mathrm{x}_0{=}\Uparrow\mathfrak{p}.\mathrm{e}.\mathrm{x}{:}\mathit{Ind}\right] \end{array}\right] . \left[\ \mathrm{e} \ : \ \mathrm{whistle}(c.\mathfrak{s}.\mathrm{x}_0)\ \right]\urcorner$$

On the basis of (41), we can create a new function with the same effect which will have the same domain and return the same results for each element in the domain but in which any dependency on '$c.\mathfrak{s}.\mathrm{x}_0$' is replaced by a dependency on '$c.\mathfrak{p}$.e.x' as in (42).

(42)
$$\ulcorner\lambda c{:}\left[\begin{array}{l} \mathit{Cntxt} \\ \mathfrak{p}{:}\left[\begin{array}{l} \mathfrak{c}^*{:}\left[\begin{array}{l} \mathit{Cntxt} \\ \mathfrak{c}{:}\left[\begin{array}{l} \mathrm{f}{:}\left[\begin{array}{l} \mathrm{f}{:}\mathit{PropCntxt} \\ \mathrm{a}{:}\mathit{PropCntxt} \end{array}\right] \\ \mathrm{a}{:}\mathit{PropCntxt} \end{array}\right] \end{array}\right] \\ \mathrm{restr}{=}\mathrm{man}'{:}\mathit{Ppty} \\ \mathrm{scope}{=}\mathrm{walk}'|_{\mathcal{F}(\mathrm{restr})}{:}\mathit{Ppty} \\ \mathrm{e}{:}\left[\begin{array}{l} \mathrm{x}{:}\mathfrak{T}(\Uparrow\mathrm{restr}) \\ \mathrm{e}{:}\mathfrak{P}(\Uparrow\mathrm{scope})\{\mathrm{x}\} \end{array}\right] \end{array}\right] \\ \mathfrak{s}{:}\left[\mathrm{x}_0{=}\Uparrow\mathfrak{p}.\mathrm{e}.\mathrm{x}{:}\mathit{Ind}\right] \end{array}\right] . \left[\ \mathrm{e} \ : \ \mathrm{whistle}(c.\mathfrak{p}.\mathrm{e}.\mathrm{x})\ \right]\urcorner$$

Since nothing now depends on the path '$\mathfrak{s}.\mathrm{x}_0$' in the context and $\mathfrak{T}(\mathrm{man}')$', that is, the type restriction on the path '$\mathfrak{p}$.e.x', is a subtype of '*Ind*', the type from which the singleton type on the path '$\mathfrak{s}.\mathrm{x}_0$' is derived, we can remove the path '$\mathfrak{s}.\mathrm{x}_0$' without changing the extension of any records that are witnesses for the context type. Thus we obtain (43).

(43) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{p}{:}\left[\begin{array}{l}\mathfrak{c}^*{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{c}{:}\left[\begin{array}{l}\text{f}{:}\left[\begin{array}{l}\text{f}{:}\textit{PropCntxt}\\ \text{a}{:}\textit{PropCntxt}\end{array}\right]\\ \text{a}{:}\textit{PropCntxt}\end{array}\right]\end{array}\right]\\ \text{restr=man}'{:}\textit{Ppty}\\ \text{scope=walk}'|_{\mathcal{F}(\text{restr})}{:}\textit{Ppty}\\ \text{e}{:}\left[\begin{array}{l}\text{x}{:}\mathfrak{T}(\Uparrow\text{restr})\\ \text{e}{:}\mathfrak{P}(\Uparrow\text{scope})\{\text{x}\}\end{array}\right]\end{array}\right]\end{array}\right] . \left[\ \text{e} \ : \ \text{whistle}(c.\mathfrak{p}.\text{e.x})\ \right]\urcorner$$

(Repeated in Appendix 17) Suppose that T is a record type and that π_1 and π_2 are paths in T. Then we use $T_{\pi_1=\pi_2}$ to represent the type exactly like T except that $T_{\pi_1=\pi_2}.\pi_1 = (T.\pi_1)_{\pi_2}$, that is, whatever type, T', is at the end of the path π_1, is replaced by the singleton type T'_{π_2}, or if $T.\pi_1$ is

$$\langle\lambda v_1{:}T_1 \ldots \lambda v_n{:}T_n \,.\, T'((v_1,\ldots,v_n)), \Pi\rangle$$

then it is replaced by

$$\langle\lambda v_1{:}T_1 \ldots \lambda v_n{:}T_n \,.\, (T'((v_1,\ldots,v_n)))_{\pi_2}, \Pi\rangle$$

We use $T_{\pi_{11}=\pi_{21},\ldots,\pi_{1n}=\pi_{2n}}$ to represent $(\ldots(T_{\pi_{11}=\pi_{21}})\ldots)_{\pi_{1n}=\pi_{2n}}$.

Suppose that φ_1 and φ_2 and parametric contents, $\pi_{11}\ldots\pi_{1n} \in \text{paths}(\varphi_1.\text{bg})$ and $\pi_{21}\ldots\pi_{2n} \in \text{paths}(\mathcal{F}_{\text{quasi}^*}(\varphi_2.\text{fg}))$, then the *content* φ_1 *given* φ_2 *with alignment of* π_{11} *and* π_{21},…,π_{1n} *and* π_{2n}, $\varphi_1 \mid_{\pi_{11},\pi_{21};\ldots;\pi_{1n},\pi_{2n}} \varphi_2$, is

$$\left[\begin{array}{lcl}\text{bg} & = & (\varphi_1.\text{bg} \wedge \left[\mathfrak{p}{:}\mathcal{F}_{\text{quasi}^*}(\varphi_2.\text{fg})\right])_{\pi_{11}=\mathfrak{p}.\pi_{21},\ldots,\pi_{1n}=\mathfrak{p}.\pi_{2n}}\\ \text{fg} & = & \lambda c{:}\text{bg} \,.\, \varphi_1(c)\end{array}\right]$$

This gives us a way of combining two parametric contents. Now we need a way of combining the kinds of parametric content types that we are using for underspecified interpretation. If T_1 and T_2 are types of parametric contents, then there is a combined type, $\mathfrak{C}(T_1, T_2)$, whose witnesses include witnesses for T_1 given a witness for T_2 with some possible alignment between the two. The witnesses of $\mathfrak{C}(T_1, T_2)$ are characterized recursively by:

1. if $\varphi : T_1$, then $\varphi : \mathfrak{C}(T_1, T_2)$

2. if $\varphi_1 : \mathfrak{C}(T_1, T_2)$,
   $\pi_{11}, \ldots, \pi_{1n} \in \mathrm{paths}(\varphi_1.\mathrm{bg})$,
   $\varphi_2 : T_2$ and
   $\pi_{21}, \ldots, \pi_{2n} \in \mathrm{paths}(\mathcal{F}_{\text{quasi}^*}(\varphi_2.\mathrm{fg}))$,

   then

$$\varphi_1|_{\pi_{11},\pi_{21};\ldots;\pi_{1n},\pi_{2n}}\varphi_2 : \mathfrak{C}(T_1, T_2)$$

What we need then for the content type of the utterance is $\mathfrak{S}(\mathfrak{C}(T_1, T_2))$. We can express this by means of the action rule in (44).

This does not express any of the linguistic constraints concerning what anaphors can be related to what antecedents.

no dog which chases a cat catches it This example is an instance of what is known in the literature as *donkey anaphora*. For a brief overview with references to a large linguistic literature see King and Lewis (2018). For good overviews up to the mid nineties from a linguistic perspective see Chierchia (1995), Chapter 2 and Kanazawa (1994). Our treatment of donkey anaphora will treat *it* in (45a) more like the kind of discourse anaphora discussed in the previous example rather than direct binding of a pronoun by a quantifier. In this way it follows the classic linguistic treatment of donkey anaphora in DRT, first formulated in Kamp (1981). Some evidence for this can be taken from (45b), where it is difficult to relate the singular pronoun *it* to *every cat*, and (45c) where the plural pronoun *them* can be related *every cat*. This follows the pattern of discourse anaphora illustrated in (45d) and (45e).

(45) a. no dog which chases a cat catches it
b. no dog which chases every cat catches it
c. no dog which chases every cat catches them
d. Every cat miaowed. It wanted milk.
e. Every cat miaowed. They wanted milk.

The key to the treatment of donkey anaphora is a process of local accommodation of context in a parametric property. Consider a parametric content for the verb-phrase *catches it* given in (46).

(46) $$\ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix}\\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\mathit{PropCntxt}\\ \mathrm{a}{:}\mathit{PropCntxt}\end{bmatrix}\end{bmatrix}\ .\ \ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\end{bmatrix}\ .\ \begin{bmatrix}\mathrm{e} & : & \mathrm{catch}^{\dagger}(r.\mathrm{x},\ c.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner\urcorner$$

(44)
$$\frac{s_{i,A} :_A \left[\text{shared:}\left[\text{latest-utterance:}\left[\text{cont:}\textit{ContType}\right]\right]\right] \qquad u^* :_A \left[\text{cont:}\textit{ContType}\right]}{s_{i+1,A} :_A \left[\text{shared:}\left[\text{latest-utterance:}\left[\text{cont=}\mathfrak{S}(\mathfrak{C}(u^*\text{.cont}, s_{i,A}\text{.shared.latest-utterance.cont})){:}\textit{ContType}\right]\right]\right]}$$

Local accommodation involves "moving" the type of the context into the domain type of the property under the label '$\mathfrak{c}$' as in (47), adjusting any paths in c addressed in the resulting function to paths in r beginning with '$\mathfrak{c}$'.

$$(47)\quad \ulcorner\lambda c{:}\textit{Cntxt}\ .\ \ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\textit{Ind} \\ \mathfrak{c}{:}\begin{bmatrix}\textit{Cntxt} \\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\end{bmatrix} \\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt} \\ \mathrm{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix}\end{bmatrix}\ .\ \begin{bmatrix}\mathrm{e} & : & \mathrm{catch}^{\dagger}(r.\mathrm{x},\, r.\mathfrak{c}.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner\urcorner$$

In general we define a localization operation, $\mathcal{L}$, on parametric properties characterized in (48).

(48) If $\mathcal{P}$ is a parametric property of the form

$$\ulcorner\lambda c{:}T_1\ .\ulcorner\lambda r{:}T_2\ .\ \varphi\urcorner\urcorner$$

then the *localization of* $\mathcal{P}$, $\mathcal{L}(\mathcal{P})$, is

$$\ulcorner\lambda c{:}\textit{Cntxt}\ .\ \ulcorner\lambda r{:}T_2 \dot{\wedge} \begin{bmatrix}\mathfrak{c}{:}T_1\end{bmatrix}\ .\ \varphi_{c.\pi\rightsquigarrow r.\mathfrak{c}.\pi}\urcorner\urcorner$$

If we use the localized content (47) as the content of the verb phrase, then, after combination with *no dog which chases a cat*, the scope of the quantifier will become (49).

$$(49)\quad \ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\textit{Ind} \\ \mathfrak{c}{:}\begin{bmatrix}\textit{Cntxt} \\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\end{bmatrix} \\ \mathfrak{c}{:}\begin{bmatrix}\mathrm{f}{:}\textit{PropCntxt} \\ \mathrm{a}{:}\textit{PropCntxt}\end{bmatrix}\end{bmatrix} \\ \mathrm{e}_1{:}\mathrm{dog}(\mathrm{x}) \\ \mathrm{e}_2{:}\begin{bmatrix}\mathrm{x}{:}\mathfrak{T}(\mathrm{cat}') \\ \mathrm{e}{:}\mathrm{chase}^{\dagger}(\Uparrow\mathrm{x},\, \mathrm{x})\end{bmatrix}\end{bmatrix}\ .\ \begin{bmatrix}\mathrm{e} & : & \mathrm{catch}^{\dagger}(r.\mathrm{x},\, r.\mathfrak{c}.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner$$

We can paraphrase this as "the property of being a dog which chases a cat and there is something which it catches". In order to obtain the anaphora we need to align the following two paths in the domain type of this function: '$\mathfrak{c}.\mathfrak{s}.\mathrm{x}_0$' and '$\mathrm{e}_2.\mathrm{x}$'. This we do in (50) by creating a manifest field on the former path.

(50) $$\ulcorner\lambda r{:}\begin{bmatrix}\text{x}:\textit{Ind}\\ \mathfrak{c}:\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}:\begin{bmatrix}\text{x}_0{=}\Uparrow^2\text{e}_2.\text{x}:\textit{Ind}\end{bmatrix}\\ \mathfrak{c}:\begin{bmatrix}\text{f}:\textit{PropCntxt}\\ \text{a}:\textit{PropCntxt}\end{bmatrix}\end{bmatrix}\\ \text{e}_1:\text{dog(x)}\\ \text{e}_2:\begin{bmatrix}\text{x}:\mathfrak{T}(\text{cat}')\\ \text{e}:\text{chase}^{\dagger}(r.\text{x},\ \text{x})\end{bmatrix}\end{bmatrix}.\ \begin{bmatrix}\text{e} & : & \text{catch}^{\dagger}(r.\text{x},\ r.\mathfrak{c}.\mathfrak{s}.\text{x}_0)\end{bmatrix}\urcorner$$

We can paraphrase this as “the property of being a dog which chases a cat and catches that cat”.

In order to include such properties with aligned paths as the scope of quantifiers in our interpretations, we will first generalize the characterization of alignment of paths given on p. 430f to functions in (51) (repeated in Appendix 17).

(51) If $\varphi = \ulcorner\lambda r{:}T\ .\ \psi\urcorner$ and $\pi_1, \pi_2 \in \text{paths}(T)$, then

$$\varphi_{\pi_1=\pi_2} = \ulcorner\lambda r{:}T_{\pi_1=\pi_2}\ .\ \psi\urcorner$$

We then add two further clauses to the characterization of witnesses for $\mathfrak{S}(T)$ for content types, T, in (52) with the new additions boxed.

(52) a. If T : *ContType*, then $\mathfrak{S}(T)$ is a type

b. The witnesses of $\mathfrak{S}(T)$ are characterized by

1. if $\varphi : T$ then $\varphi : \mathfrak{S}(T)$
2. $\boxed{\text{if } \varphi : \mathfrak{S}(T) \text{ and } \varphi : \textit{PPpty}\text{, then } \mathcal{L}(\varphi) : \mathfrak{S}(T)}$
3. $\boxed{\text{if } \varphi : \mathfrak{S}(T),\ \varphi \sqsubseteq \begin{bmatrix}\text{scope}{=}\psi{:}\textit{Ppty}\end{bmatrix} \text{ and } \pi_1, \pi_2 \in \text{paths}(\psi.\text{bg}), \text{ then } \varphi[\text{scope} = \psi_{\pi_1=\pi_2} : \textit{Ppty}] : \mathfrak{S}(T)}$
4. if $\alpha\mathcal{O}\beta : \mathfrak{S}(T)$, (for some combination operation, $\mathcal{O}$) and $\alpha\mathcal{O}_{i,j}\beta$ is defined (for some natural numbers, i and j), then $\alpha\mathcal{O}_{i,j}\beta : \mathfrak{S}(T)$
5. if $\varphi : \mathfrak{S}(T)$ and φ is in the range of ‘storage’, then $\text{storage}(\varphi) : \mathfrak{S}(T)$
6. if $\varphi : \mathfrak{S}(T)$ and ‘x_i’ and φ are appropriate arguments to ‘retrieve’, then $\text{retrieve}(\text{x}_i, \varphi) : \mathfrak{S}(T)$
7. nothing is a witness for $\mathfrak{S}(T)$ except as required above.

Overall we can get a parametric content for *no dog which chases a cat catches it* which looks like (53) where **dog⌢which⌢chases⌢a⌢cat** and **catches⌢it** represent parametric contents which have not undergone operations introduced by $\mathfrak{S}$.

(53) λc:*Cntxt* .

$$\begin{bmatrix} \text{restr}=\textbf{dog}^\frown\textbf{which}^\frown\textbf{chases}^\frown\textbf{a}^\frown\textbf{cat}(c) & : & \mathit{Ppty} \\ \text{scope}=(\mathcal{L}(\textbf{catches}^\frown\textbf{it})(c)|_{\mathcal{F}(\textbf{dog}^\frown\textbf{which}^\frown\textbf{chases}^\frown\textbf{a}^\frown\textbf{cat})})_{\text{c.s.x}_0=\text{e}_2.\text{x}} & : & \mathit{Ppty} \\ \text{e:no(restr, scope)} & & \end{bmatrix}$$

Given the semantics we have specified for the determiner *no*, (53) is (54).

(54) λc:*Cntxt* .

$$\begin{bmatrix} \text{restr}=\textbf{dog}^\frown\textbf{which}^\frown\textbf{chases}^\frown\textbf{a}^\frown\textbf{cat}(c) & : & \mathit{Ppty} \\ \text{scope}=(\mathcal{L}(\textbf{catches}^\frown\textbf{it})(c)|_{\mathcal{F}(\textbf{dog}^\frown\textbf{which}^\frown\textbf{chases}^\frown\textbf{a}^\frown\textbf{cat})})_{\text{c.s.x}_0=\text{e}_2.\text{x}} & : & \mathit{Ppty} \\ \text{e:}\begin{bmatrix} \text{X} & : & \text{every}^{w}(\text{restr}) \\ \text{f} & : & ((x : \mathfrak{T}(X)) \rightarrow \neg\mathfrak{P}(\text{scope})\{x\}) \end{bmatrix} & & \end{bmatrix}$$

We can paraphrase this content as “for every dog which chases a cat it’s not the case that it’s a dog which chases a cat$_i$ and catches it$_i$” where the subscript on *cat* and *it* indicates that *it* is anaphorically related to *a cat*.

This treatment of donkey anaphora is essentially similar to that of Chierchia (1995) in that it associates an existential reading the pronoun it, that is, it is not the case that there is a cat which the dog chases that it also catches. The dog does not catch any of the cats it chases. If we use *every* instead of *no* we get a reading which is paraphrased as “every dog which chases a cat is a dog which chases a cat and catches it”. This is what is known in the literature as a *weak reading* or a *∃-reading*. It says that every dog which chases a cat catches *some* cat that it chases (but not necessarily all). This reading may intuitively not be appropriate for *every dog that chases a cat catches it* which for many speakers would suggest that the dogs catch all the cats they chase. However, the weak reading is important for the examples in (55).

(55) a. Every person who had a dime put it in the parking meter. (Pelletier and Schubert, 1989)

b. Every man who has a daughter thinks she is the most beautiful girl in the world (Cooper, 1979)

(55a) does not seem to suggest that anybody who had several dimes put them all in the meter and (55b) does not seem to commit a man who has two daughters to believe the contradictory proposition that they are both the one and only most beautiful girl in the world. What then do we say about the original donkey sentences like *every farmer who owns a donkey likes it* which were analyzed by Geach and the classical analyses in DRT and type theory as having a strong reading in which every farmer who owns a donkey likes any donkey that he owns? One option is to say that we only need the weak reading for such sentences as it is consistent with the stronger reading. Many speakers feel that it unclear what the sentence means if some man owns more than one donkey. That is, *every farmer who owns a donkey likes it* requires that for every donkey owning farmer there is at least one donkey the farmer owns such that she likes it. This allows for the farmers to like all their donkeys but does not require it. (See Kanazawa, 1994 for a discussion of this.) In the case of *every dog which chases a cat catches it* I have the following intuition:

> if there is a dog under consideration which is involved in two distinct cat-chasing events (with a single cat) and only succeeds in catching the cat in one of the two events, then this seems to make the sentence false;
>
> if there is a dog under consideration which is involved in a single event of chasing two cats and only succeeds in catching one of the cats in that event, then it seems that the sentence could still be true.

This suggests an analysis which requires that every relevant cat-chasing event must involve the catching of at least one of the cats chased in that event. Firm judgements concerning such intuitions are notoriously hard to come by. Chierchia (1995) argues that the strong $\forall$-reading is necessary because of examples like (56).

(56) Every man who owned a slave owned his offspring

(56) is a modification of an example in Heim (1990) which is used to make a different argument. It seems like a single instance of somebody owning a slave but not the slave's offspring would be sufficient to falsify (56). Though again, in the case of a slave who has several offspring one of which is owned by somebody else, it seems to me that it is unclear whether that is sufficient to falsify the sentence.

One option is to recreate the original Geach reading by using the variant of the purification operation, '$\mathfrak{P}^{\forall}$', in (13), p. 355 which introduces a function on local contexts. Using '$\mathfrak{P}^{\forall}$' instead of '$\mathfrak{P}$' will have the consequence that the content of *every farmer who owns a donkey likes it* will be (57).

(57) $\lambda c{:}\mathit{Cntxt}\ .$
$$\left[\begin{array}{lll}\text{restr}{=}\textbf{farmer}^\frown\textbf{who}^\frown\textbf{owns}^\frown\textbf{a}^\frown\textbf{donkey}(c) & : & \mathit{Ppty}\\ \text{scope}{=}(\mathcal{L}(\textbf{likes}^\frown\textbf{it})(c)|_{\mathcal{F}(\textbf{farmer}^\frown\textbf{who}^\frown\textbf{owns}^\frown\textbf{a}^\frown\textbf{donkey})})_{\text{c.s.x}_0=\text{e}_2\text{.x}} & : & \mathit{Ppty}\\ \text{e:every(restr, scope)} & & \end{array}\right]$$

Given the general witness condition associated with 'every', (57) is equivalent to (58).

(58) $\lambda c{:}\mathit{Cntxt}\ .$
$$\begin{bmatrix} \text{restr}{=}\textbf{farmer}^\frown\textbf{who}^\frown\textbf{owns}^\frown\textbf{a}^\frown\textbf{donkey}(c) & : & \mathit{Ppty} \\ \text{scope}{=}(\mathcal{L}(\textbf{likes}^\frown\textbf{it})(c)|_{\mathcal{F}(\textbf{farmer}^\frown\textbf{who}^\frown\textbf{owns}^\frown\textbf{a}^\frown\textbf{donkey})})_{c.\mathfrak{s}.\mathrm{x}_0=\mathrm{e}_2.\mathrm{x}} & : & \mathit{Ppty} \\ \text{e}{:}\begin{bmatrix} \mathrm{X} & : & \text{every}^w(\text{restr}) \\ \mathrm{f} & : & ((x:\mathfrak{T}(X)) \to \mathfrak{P}(\text{scope})\{x\}) \end{bmatrix} & & \end{bmatrix}$$

If we use a variant of this witness condition which uses '$\mathfrak{P}^{\forall}$' instead of '$\mathfrak{P}$' we have (59).

(59) $\lambda c{:}\mathit{Cntxt}\ .$
$$\begin{bmatrix} \text{restr}{=}\textbf{farmer}^\frown\textbf{who}^\frown\textbf{owns}^\frown\textbf{a}^\frown\textbf{donkey}(c) & : & \mathit{Ppty} \\ \text{scope}{=}(\mathcal{L}(\textbf{likes}^\frown\textbf{it})(c)|_{\mathcal{F}(\textbf{farmer}^\frown\textbf{who}^\frown\textbf{owns}^\frown\textbf{a}^\frown\textbf{donkey})})_{c.\mathfrak{s}.\mathrm{x}_0=\mathrm{e}_2.\mathrm{x}} & : & \mathit{Ppty} \\ \text{e}{:}\begin{bmatrix} \mathrm{X} & : & \text{every}^w(\text{restr}) \\ \mathrm{f} & : & ((x:\mathfrak{T}(X)) \to \mathfrak{P}^{\forall}(\text{scope})\{x\}) \end{bmatrix} & & \end{bmatrix}$$

Let us check that (59) does in fact give us the strong $\forall$-reading. We will do this by showing that the value associated with the label 'scope' will be paraphrasable as "the property of being an individual such that if it's a farmer who owns a donkey, she likes that donkey", that is, likes every donkey she owns. We can show this by unpacking the expression representing the scope in (59). '**likes**$^\frown$**it**', according to our treatment of free pronouns and extensional verbs, will be (60).

(60) $$\ulcorner\lambda c{:}\begin{bmatrix}\text{Cntxt} \\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix}\end{bmatrix} . \ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & \text{like}^{\dagger}(r.\mathrm{x}, c.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner\urcorner$$

Using the definition of the localization operation, $\mathcal{L}$, in (48), we can see that '$\mathcal{L}$(**likes**$^\frown$**it**)' is (61).

(61) $$\ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind} \\ \mathfrak{c}{:}\begin{bmatrix}\mathit{Cntxt} \\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & \text{like}^{\dagger}(r.\mathrm{x}, r.\mathfrak{c}.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner\urcorner$$

Applying (61) to any context, c, will obtain (62).

(62) $$\ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind} \\ \mathfrak{c}{:}\begin{bmatrix}\mathit{Cntxt} \\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & \text{like}^{\dagger}(r.\mathrm{x}, r.\mathfrak{c}.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner$$

Restricting (62) by '$\mathcal{F}$(**farmer**$^\frown$**who**$^\frown$**owns**$^\frown$**a**$^\frown$**donkey**)', yields (63).

$$(63)\quad \ulcorner\lambda r{:}\left[\begin{array}{l} \text{x}:\mathit{Ind}\\ \mathfrak{c}:\left[\begin{array}{l}\mathit{Cntxt}\\ \mathfrak{s}:\left[\text{x}_0:\mathit{Ind}\right]\\ \mathfrak{c}:\left[\begin{array}{l}\text{f}:\mathit{PropCntxt}\\ \text{a}:\mathit{PropCntxt}\end{array}\right]\end{array}\right]\\ \text{e}_1:\text{farmer(x)}\\ \text{e}_2:\left[\begin{array}{l}\text{x}:\mathfrak{T}(\text{donkey}')\\ \text{e}:\text{own}^\dagger(\Uparrow\text{x, x})\end{array}\right]\end{array}\right] . \left[\ \text{e} \ :\ \text{like}^\dagger(r.\text{x}, r.\mathfrak{c}.\mathfrak{s}.\text{x}_0)\ \right]\urcorner$$

Aligning the paths '$\mathfrak{c}.\mathfrak{s}.\text{x}_0$' and '$\text{e}_2.\text{x}$' in (63) yields (64).

$$(64)\quad \ulcorner\lambda r{:}\left[\begin{array}{l} \text{x}:\mathit{Ind}\\ \mathfrak{c}:\left[\begin{array}{l}\mathit{Cntxt}\\ \mathfrak{s}:\left[\text{x}_0{=}\Uparrow^2\text{e}_2.\text{x}:\mathit{Ind}\right]\\ \mathfrak{c}:\left[\begin{array}{l}\text{f}:\mathit{PropCntxt}\\ \text{a}:\mathit{PropCntxt}\end{array}\right]\end{array}\right]\\ \text{e}_1:\text{farmer(x)}\\ \text{e}_2:\left[\begin{array}{l}\text{x}:\mathfrak{T}(\text{donkey}')\\ \text{e}:\text{own}^\dagger(\Uparrow\text{x, x})\end{array}\right]\end{array}\right] . \left[\ \text{e} \ :\ \text{like}^\dagger(r.\text{x}, r.\mathfrak{c}.\mathfrak{s}.\text{x}_0)\ \right]\urcorner$$

Finally, in the 'e.f'-field in (59) '$\mathfrak{P}^\forall$' is applied to the scope, that is, (64). Thus '$\mathfrak{P}^\forall$(scope)' represents (65).

$$(65)\quad \ulcorner\lambda r{:}\left[\text{x}:\mathit{Ind}\right] . ((r'{:}\left[\begin{array}{l} \text{x}{=}r.\text{x}:\mathit{Ind}\\ \mathfrak{c}:\left[\begin{array}{l}\mathit{Cntxt}\\ \mathfrak{s}:\left[\text{x}_0{=}\Uparrow^2\text{e}_2.\text{x}:\mathit{Ind}\right]\\ \mathfrak{c}:\left[\begin{array}{l}\text{f}:\mathit{PropCntxt}\\ \text{a}:\mathit{PropCntxt}\end{array}\right]\end{array}\right]\\ \text{e}_1:\text{farmer(x)}\\ \text{e}_2:\left[\begin{array}{l}\text{x}:\mathfrak{T}(\text{donkey}')\\ \text{e}:\text{own}^\dagger(\Uparrow\text{x, x})\end{array}\right]\end{array}\right])\rightarrow \left[\ \text{e} \ :\ \text{like}^\dagger(r'.\text{x}, r'.\mathfrak{c}.\mathfrak{s}.\text{x}_0)\ \right])\urcorner$$

(65) can be paraphrased as "the property of being an individual, x, if x is a farmer and owns a donkey, y, then x likes y" thus requiring that every farmer likes all of the donkeys she owns. The whole sentence says that every farmer who owns a donkey has this property.

The domain of the function introduced is farmers who own a donkey in our analysis of both the weak and the strong readings. This is in contrast to the classical treatment of the strong reading (Geach, 1962; Kamp and Reyle, 1993) which can be construed as quantification over pairs of farmers and donkeys. This works in the case of universal quantification. The sentence *every* *farmer who owns a donkey likes it* can be construed as “every farmer-donkey pair such that the farmer owns the donkey is such that the farmer likes the donkey”. However, for other generalized quantifiers, such as *most* this represents an incorrect paraphrase. Thus *most farmers who own a donkey like it* cannot be construed as “most farmer-donkey pairs such that the farmer owns the donkey are such that the farmer likes the donkey”. Chierchia (1995) gives a good account of why this is so. Suppose we have five farmers four of which have exactly one donkey and do not like it. The fifth farmer has fifty donkeys and likes them all. Clearly in this case most farmer-donkey pairs are such that the farmer likes the donkey but it is not the case that most farmers who own a donkey like it. This problem is known in the literature as the *proportion problem*. The treatment that we have proposed here does not suffer from it since the quantification is over farmers who own a donkey rather than farmer-donkey pairs.

Sam likes him/himself The treatment of anaphora that we have presented so far overgenerates in that it does not take account of the restrictions on anaphoric possibilities which exist in natural languages. For example, the sentence (66) does not allow a reading in which *him* is anaphorically related to *Sam*.

(66) Sam likes him

Note that this is not the same as saying that *Sam* and *him* are not allowed to refer to the same individual. There are cases where this is possible, though often bizarre. Suppose that Sam finds a wikipedia page describing a person with interests and achievements very similar to his own but he does not realize that in fact somebody has written a page about Sam himself. Sam can decide that he likes the person described in the entry and this can be described by (66). The basic technique that we employ for making this distinction is illustrated by the types in (67).

(67) a. $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c} & : & \text{named(x, “Sam”)} \\ \text{y} & : & \textit{Ind} \\ \text{e} & : & \text{like(x, y)} \end{bmatrix}$

b. $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c} & : & \text{named(x, “Sam”)} \\ \text{y=x} & : & \textit{Ind} \\ \text{e} & : & \text{like(x, y)} \end{bmatrix}$

c. $\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c} & : & \text{named(x, “Sam”)} \\ \text{e} & : & \text{like(x, x)} \end{bmatrix}$

In (67a) we have two fields for individuals labelled by ‘x’ and ‘y’ respectively. There is nothing to prevent these fields from being filled by the same individual in some of the witnesses for the type. In (67b), however, the fields will be filled by the same individual in any witness for the type. A further possibility is to have just one field for an individual as in (67c). Let us see how these options play out in actual parametric contents for *Sam likes him* resulting currently in a content where *him* is anaphorically related to *Sam*. A parametric content for *likes him* is (68), parallel to contents for similar verb phrases we have seen previously.

(68) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\end{bmatrix}\\ \mathfrak{c}{:}\left[\begin{array}{l}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{array}\right]\end{array}\right] . \ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\textit{Ind}\end{bmatrix} . \begin{bmatrix}\mathrm{e} & : & \mathrm{like}^{\dagger}(r.\mathrm{x}, c.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner\urcorner$$

Using the lexical rule for proper nouns we obtain the parametric content (69) for *Sam*.

(69) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathfrak{c}{:}\left[\begin{array}{l}\mathrm{x}{:}\textit{Ind}\\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Sam”})\end{array}\right]\end{array}\right] . \lambda P{:}\textit{Ppty} . P\{c.\mathfrak{c}.\mathrm{x}\}\urcorner$$

Let us represent (69) as **Sam**. Using ‘storage’ we obtain an additioinal parametric content given in (70).

(70) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathrm{q}{:}\begin{bmatrix}\mathrm{x}_0{=}\textbf{Sam}{:}\textit{PQuant}\end{bmatrix}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\end{bmatrix}\end{array}\right] . \lambda P{:}\textit{Ppty} . P\{c.\mathfrak{s}.\mathrm{x}_0\}\urcorner$$

We can now obtain a parametric content for *Sam likes him* by using (70) and (68) combined with ‘$@_{0,1}$’. This is given in (71).

(71) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\textit{Cntxt}\\ \mathrm{q}{:}\begin{bmatrix}\mathrm{x}_0{=}\textbf{Sam}{:}\textit{PQuant}\end{bmatrix}\\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\textit{Ind}\end{bmatrix}\\ \mathfrak{c}{:}\left[\begin{array}{l}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{array}\right]\end{array}\right] . \begin{bmatrix}\mathrm{e} & : & \mathrm{like}^{\dagger}(c.\mathfrak{s}.\mathrm{x}_0, c.\mathfrak{s}.\mathrm{x}_0)\end{bmatrix}\urcorner$$

We can now apply ‘retrieve’ to ‘x_0’ and (71) to obtain (72).

(72) $$\ulcorner\lambda c{:}\left[\begin{array}{l}\mathfrak{c}{:}\left[\begin{array}{l}\mathrm{f}{:}\left[\begin{array}{l}\mathrm{x}{:}\textit{Ind}\\ \mathrm{e}{:}\mathrm{named}(\mathrm{x}, \text{“Sam”})\end{array}\right]\\ \mathrm{a}{:}\left[\begin{array}{l}\mathrm{f}{:}\textit{PropCntxt}\\ \mathrm{a}{:}\textit{PropCntxt}\end{array}\right]\end{array}\right]\end{array}\right] . \begin{bmatrix}\mathrm{e} & : & \mathrm{like}^{\dagger}(c.\mathfrak{c}.\mathrm{f}.\mathrm{x}, c.\mathfrak{c}.\mathrm{f}.\mathrm{x})\end{bmatrix}\urcorner$$

(72) is not an appropriate content for *Sam likes him* although it would be appropriate for *Sam likes himself*. Simplifying a good deal, pronouns which are not reflexive (like *himself*) cannot be anaphorically related to an antecedent within the same clause. This is Principle B of Chomsky's binding theory (Chomsky, 1981). We shall treat this by adding a field labelled '$\mathfrak{l}$' in the context requiring an assignment which keeps track of pronouns which are local. We adjust the definition of *Cntxt* to be (73).

$$(73)\quad \begin{bmatrix} \mathfrak{q} & : & \mathit{QStore} \\ \mathfrak{s} & : & \mathit{Assgnmnt} \\ \mathfrak{l} & : & \mathit{Assgnmnt} \\ \mathfrak{w} & : & \mathit{Assgnmnt} \\ \mathfrak{g} & : & \mathit{Assgnmnt} \\ \mathfrak{c} & : & \mathit{PropCntxt} \end{bmatrix}$$

The generating parametric content for *he* is (74) where we mark 'x_0' in the $\mathfrak{l}$-field.

$$(74)\quad \ulcorner\lambda c{:}\begin{bmatrix} \mathit{Cntxt} \\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix} \\ \mathfrak{l}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix} \end{bmatrix} .\ \lambda P{:}\mathit{Ppty}\ .\ P\{c.\mathfrak{s}.\mathrm{x}_0\}\urcorner$$

We adjust (27) to include reference to paths '$\mathfrak{l}.\mathrm{x}_i$' by adding the boxed material as in (75).

(75) If $\alpha : \begin{bmatrix}\mathrm{bg}{:}\mathit{CntxtType} \\ \mathrm{fg}{:}(\mathrm{bg}{\rightarrow}(T_1 \rightarrow T_2))\end{bmatrix}$ and $\beta : \begin{bmatrix}\mathrm{bg}{:}\mathit{CntxtType} \\ \mathrm{fg}{:}(\mathrm{bg}{\rightarrow}\ T_1)\end{bmatrix}$ and $\alpha.\mathrm{bg} \sqsubseteq \begin{bmatrix}\mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_i{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$ and $\mathrm{incr}(\beta.\mathrm{bg}, \alpha.\mathrm{bg}) \sqsubseteq \begin{bmatrix}\mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_j{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$ but $\mathrm{incr}(\beta.\mathrm{bg}, \alpha.\mathrm{bg}) \not\sqsubseteq \begin{bmatrix}\mathfrak{q}{:}\begin{bmatrix}\mathrm{x}_j{:}\mathit{PQuant}\end{bmatrix}\end{bmatrix}$ $\boxed{\text{and } \mathrm{incr}(\beta.\mathrm{bg}, \alpha.\mathrm{bg}) \not\sqsubseteq \begin{bmatrix}\mathfrak{l}{:}\begin{bmatrix}\mathrm{x}_j{:}\mathit{PQuant}\end{bmatrix}\end{bmatrix}}$, then the *combination of* α *and* β *based on functional application and anaphoric relation of* j *to* i, $\alpha@_{i,j}\beta$, is

$$\ulcorner\lambda c{:}[\alpha.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}} \mathbin{\dot\wedge} [\mathrm{incr}([\beta.\mathrm{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}, \alpha.\mathrm{bg})]_{\mathfrak{s}.\mathrm{x}_j\rightsquigarrow\mathfrak{s}.\mathrm{x}_i}\ .$$
$$[\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{f}}(c)([\mathrm{incr}([\beta.\mathrm{fg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathrm{a}}, \alpha.\mathrm{bg})]_{\mathfrak{s}.\mathrm{x}_j\rightsquigarrow\mathfrak{s}.\mathrm{x}_i}(c))\urcorner$$

The information about locality represented by the '$\mathfrak{l}$'-field will now percolate up as a constraint on the context as we combine consituents. However, once we reach a sentence we no longer want the pronoun to count as local since pronouns can be anaphorically related to antecedents outside the clause in which they occur as in *Sam*$_i$ *thinks that she*$_i$ *is lucky*. We define an operation B on parametric contents (think of Principle "B" or "Boundary") which uses asymmetric merge to remove the locality constraint. B is defined in (76).

(76) If α is a parametric content,

$$\ulcorner\lambda c{:}T\ .\ \varphi(\!(c)\!)\urcorner$$

then $B(\alpha)$ is

$$\ulcorner\lambda c{:}T\boxed{\wedge}\big[\mathfrak{l}{:}\textit{Assgnmnt}\big]\ .\ \varphi(\!(c)\!)\urcorner$$[1]

Suppose that T_1 and T_2 are of type *ContType* and that $\mathcal{O}$ is a combination operation such as @ etc., then we say that $T_1\mathcal{O}^{\mathfrak{S},B}T_2$ is also a type with the witness condition in (77).

(77) If $\alpha : T_1$, $\beta : T_2$ and $\alpha\mathcal{O}\beta$ is defined, then $B(\alpha\mathcal{O}\beta) : T_1\mathcal{O}^{\mathfrak{S},B}T_2$. Nothing else is a witness for $T_1\mathcal{O}^{\mathfrak{S},B}T_2$.

In (78) we introduce versions of 'ContForwardApp' operations which involve B.

(78) If $\mathcal{O}$ is a combination operator, then $\text{ContForwardApp}_{\mathfrak{S},\mathcal{O},B}$ is

$$\lambda u{:}\big[\text{cont:}\textit{ContType}\big]^\frown\big[\text{cont:}\textit{ContType}\big]\ .\ \big[\text{cont:}\mathfrak{S}(u[0].\text{cont}\mathcal{O}^{\mathfrak{S},B}u[1].\text{cont})\big]$$

We then introduce a notation for constituent structure rules involving locality boundaries as in (79).

(79) If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and $\mathcal{O}$ is a combination operation, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ B(T'_{\text{daughter}_1}({}_{\mathcal{O}}T'_{\text{daughter}_2}))$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \underset{\cdot}{\wedge}\ \text{ContForwardApp}_{\mathfrak{S},\mathcal{O},B}$$

Thus for example we can formulate the constituent rule that says that a sentence can consist of a noun phrase followed by a verb phrase as (80).

(80) $S \longrightarrow NP\ VP \mid B(NP'({}_{@}VP'))$

[1]This assumes that paremtric contents are defined in such as way that φ will not depend on $c.\mathfrak{l}.\pi$ for any π. Otherwise $B(\alpha)$ would need to be

$$\ulcorner\lambda c{:}T\boxed{\wedge}\big[\mathfrak{l}{:}\textit{Assgnmnt}\big]\ .\ [\varphi(\!(c)\!)]_{c.\mathfrak{l}.\pi\rightsquigarrow c.\mathfrak{s}.\pi}\urcorner$$

Reflexive pronouns, like *himself*, obey an almost complementary principle to non-reflexive pronouns – they must be anaphorically related to a local antecedent and cannot be related to a non-local antecedent. In addition, they must be related to some antecedent. They cannot simply refer to something in the context as ordinary pronouns can. In Chomsky's binding theory such pronouns are called "anaphors" and include both reflexive pronouns and reciprocals such as *each other*. In order to handle such requirements on local anaphora we will add another component to the context type under the label $\mathfrak{r}$ ("reflexive"). Thus the type *Cntxt* will now be defined as in (81).

(81) $$\begin{bmatrix} \mathfrak{q} & : & \mathit{QStore} \\ \mathfrak{s} & : & \mathit{Assgnmnt} \\ \mathfrak{l} & : & \mathit{Assgnmnt} \\ \mathfrak{r} & : & \mathit{Assgnmnt} \\ \mathfrak{w} & : & \mathit{Assgnmnt} \\ \mathfrak{g} & : & \mathit{Assgnmnt} \\ \mathfrak{c} & : & \mathit{PropCntxt} \end{bmatrix}$$

The generating parametric content for *himself* is (82) where we mark 'x_0' in the $\mathfrak{r}$-field.

(82) $$\ulcorner\lambda c{:}\begin{bmatrix} \mathit{Cntxt} \\ \mathfrak{s}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix} \\ \mathfrak{r}{:}\begin{bmatrix}\mathrm{x}_0{:}\mathit{Ind}\end{bmatrix} \end{bmatrix} \;.\; \lambda P{:}\mathit{Ppty} \;.\; P\{c.\mathfrak{s}.\mathrm{x}_0\}\urcorner$$

The reflexive marking in the context will percolate up to contexts associated with higher phrases. We provide a mechanism for removing the marking in properties and simultaneously binding the reflexive pronoun. This is given in (83).

(83) If $\mathcal{P}$ is a parametric property of the form

$$\ulcorner\lambda c{:}T_1 \;.\; \ulcorner\lambda r{:}T_2(\!(T_1)\!) \;.\; \varphi(\!(T_1,T_2)\!)\urcorner\urcorner$$

where for some natural number i, $T_1 \sqsubseteq \begin{bmatrix}\mathfrak{r}{:}\begin{bmatrix}\mathrm{x}_i{:}\mathit{Ind}\end{bmatrix}\end{bmatrix}$, then *the reflexivization of* $\mathcal{P}$, $\mathfrak{R}(\mathcal{P})$, is

$$\ulcorner\lambda c{:}(T_1\boxed{\wedge}\begin{bmatrix}\mathfrak{r}{:}\mathit{Assgnmnt}\end{bmatrix})\ominus\mathfrak{s}.\mathrm{x}_i \;.\; \ulcorner\lambda r{:}T_2 \;.\; [\varphi]_{c.\mathfrak{s}.\mathrm{x}_i\rightsquigarrow r.\mathrm{x}}\urcorner\urcorner$$

This operation removes the reflexive marking in the $\mathfrak{r}$-field of the context using asymmetric merge and also removes the corresponding path '$\mathfrak{s}.\mathrm{x}_i$' from the context type so there is no dependence on an individual labelled 'x_i' in the context. Any dependence on '$\mathfrak{s}.\mathrm{x}_i$' in the body of the property represented by φ is replaced by a dependence on 'x' in the domain of the property — note that the domain type of the property, T_2, is guaranteed to be a record type with 'x' among its labels by the requirement that $\mathcal{P}$ is a parametric property.

The operation, $\mathfrak{R}$, gives us a way of binding reflexive pronouns with verb phrases by the subject of the sentence. It does not, however, allow us to have a reflexive bound within a verb-phrase as in examples like *The guru revealed Kim$_i$ to himself$_i$*. Nor does it correctly require that both occurrences of *himself* have to be anaphorically related to *the guru* in *The guru revealed himself to himself*.[2] We shall not deal with such cases here.

We will, however, introduce a mechanism for requiring that the reflexive pronoun must be anaphorically related to something. It cannot be left as free and dependent on the context like a regular pronoun. Our strategy for doing this involves mapping a parametric content type to a subtype which excludes witnesses which have a reflexive marking involving the label $\mathfrak{r}$ in the context type. For any parametric content type, T, we characterize a subtype, $\mathfrak{A}(T)$, which excludes parametric contents with free reflexives as indicated by the $\mathfrak{r}$-field ($\mathfrak{A}$ for "Principle A"). This is characterized in (84).

(84) If T is a parametric content type, then $\mathfrak{A}(T)$ is also a parametric content type.
$\varphi : \mathfrak{A}(T)$ iff $\varphi : T$ and $\varphi.\text{bg} \not\sqsubseteq \left[\mathfrak{r}{:}\left[\text{x}_i{:}\textit{Ind}\right]\right]$, for any natural number i.

In (85) we introduce versions of 'ContForwardApp' operations which involve $\mathfrak{A}$.

(85) If $\mathcal{O}$ is a combination operator, then $\text{ContForwardApp}_{\mathfrak{S},\mathcal{O},\mathfrak{A}}$ is

$$\lambda u{:}\left[\text{cont:}\textit{ContType}\right]^\frown\left[\text{cont:}\textit{ContType}\right]\ .\ \left[\text{cont:}\mathfrak{S}(\mathfrak{A}(u[0].\text{cont}\mathcal{O}^{\mathfrak{S}}u[1].\text{cont}))\right]$$

We then introduce a notation for constituent structure rules involving locality boundaries as in (86).

(86) If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and $\mathcal{O}$ is a combination operation, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ \mathfrak{A}(T'_{\text{daughter}_1}(\mathcal{O}T'_{\text{daughter}_2}))$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2} \mathbin{\overset{\wedge}{\underset{\cdot}{}}} \text{ContForwardApp}_{\mathfrak{S},\mathcal{O},\mathfrak{A}}$$

[2]In other languages such as German and Scandinavian languages, these examples involve a different reflexive construction. (See, for example, Hellan, 1986.)

Thus for example we can formulate the constituent rule that says that a sentence can consist of a noun phrase followed by a verb phrase as (87).

(87) *VP* $\longrightarrow$ *V NP* | $\mathfrak{A}$(*V*′($_{@}$*NP*′))

Introducing the $\mathfrak{A}$-locality boundary at the VP-level as in (87) will ensure that all reflexives will be anaphorically related within a clause.

In order to include reflexives we need to extend our characterization of $\mathfrak{S}$ to include reflexive parametric contents as indicated by the boxed text in (88).

(88) a. If T : *ContType*, then $\mathfrak{S}(T)$ is a type

b. The witnesses of $\mathfrak{S}(T)$ are characterized by

1. if $\varphi : T$ then $\varphi : \mathfrak{S}(T)$
2. if $\varphi : \mathfrak{S}(T)$ and φ : *PPpty*, then $\mathcal{L}(\varphi) : \mathfrak{S}(T)$
3. $\boxed{\text{if } \varphi : \mathfrak{S}(T) \text{ and } \varphi : \textit{PPpty}\text{, then } \mathfrak{R}(\varphi) : \mathfrak{S}(T)}$
4. if $\varphi : \mathfrak{S}(T)$, $\varphi \sqsubseteq \left[\text{scope}{=}\psi{:}\textit{Ppty}\right]$ and $\pi_1, \pi_2 \in \text{paths}(\psi.\text{bg})$, then $\varphi[\text{scope} = \psi_{\pi_1=\pi_2} : \textit{Ppty}] : \mathfrak{S}(T)$
5. if $\alpha\mathcal{O}\beta : \mathfrak{S}(T)$, (for some combination operation, $\mathcal{O}$) and $\alpha\mathcal{O}_{i,j}\beta$ is defined (for some natural numbers, i and j), then $\alpha\mathcal{O}_{i,j}\beta : \mathfrak{S}(T)$
6. if $\varphi : \mathfrak{S}(T)$ and φ is in the range of ‘storage’, then $\text{storage}(\varphi) : \mathfrak{S}(T)$
7. if $\varphi : \mathfrak{S}(T)$ and ‘x_i’ and φ are appropriate arguments to ‘retrieve’, then $\text{retrieve}(x_i, \varphi) : \mathfrak{S}(T)$
8. nothing is a witness for $\mathfrak{S}(T)$ except as required above.

This is only the beginning of a theory of reflexives in English. For example, it will not on its own prevent reflexives occuring in subject position as in **Himself saw Sam* or **Kim feels that herself is welcome*. Perhaps this is simply a matter of case, which we have not treated here. There is just no nominative version of the reflexive which can be used as the subject of a tensed sentence. This is perhaps suggested by the acceptability of *Kim feels herself to be welcome* which seems to express the same content. A further example, well-known from the literature, concerns what are known as picture noun-phrases. While this treatment will allow for a correct interpretation of *Kim found a picture of herself* where *Kim* is the antecedent of *herself* it will not correctly account for *Kim found Sam’s picture of herself* where the only possible antecedent for *herself* is *Sam*. We will leave a more complete treatment of reflexives for future exploration.

8.4 Summary of resources introduced

Items that are new since Chapter 7 are marked "**New!**" and items that have been revised since Chapter 7 are marked "**Revised!**".

8.4.1 Universal grammar resources

8.4.1.1 Types

$$\mathit{Loc} \text{ — } \begin{bmatrix} \text{x-coord} & : & \mathit{Real} \\ \text{y-coord} & : & \mathit{Real} \\ \text{z-coord} & : & \mathit{Real} \end{bmatrix}$$

Phon — a basic type

e : *Phon* iff e is a phonological event

$$\mathit{SEvent} \text{ — } \begin{bmatrix} \text{e-loc} & : & \mathit{Loc} \\ \text{sp} & : & \mathit{Ind} \\ \text{au} & : & \mathit{Ind} \\ \text{e} & : & \mathit{Phon} \\ \text{c}_{\text{loc}} & : & \text{loc(e,e-loc)} \\ \text{c}_{\text{sp}} & : & \text{speaker(e,sp)} \\ \text{c}_{\text{au}} & : & \text{audience(e,au)} \end{bmatrix} \text{ (as in Chapter 2)}$$

Assgnmnt — a basic type

r : *Assgnmnt* iff r : *Rec* and $\text{labels}(r) \subset \{\text{x}_0, \text{x}_1, \ldots\}$

If T is *Assgnmnt*$\wedge T'$, T' is a record type and $\ell \in \text{labels}(T')$, then

1. if $\text{labels}(T') = \{\ell\}$, $T \ominus \ell =$ *Assgnmnt*
2. otherwise, $T \ominus \ell = \mathit{Assgnmnt} \wedge (T' \ominus \ell / T)$

PropCntxt — a basic type

r : *PropCntxt* iff r : *Rec* and $\text{labels}(r) \cap \{\text{x}_0, \text{x}_1, \ldots\} = \emptyset$

If T is *PropCntxt*$\wedge T'$ and $\pi \in \text{tpaths}(T')$ then

1. if π is ℓ and $\text{labels}(T') = \{\ell\}$, then $T \ominus \pi = \mathit{PropCntxt}$
2. otherwise, $T \ominus \pi = \mathit{PropCntxt} \wedge (T' \ominus \pi / T')$

Cntxt **Revised!** — $\begin{bmatrix} \mathfrak{q} & : & \textit{QStore} \\ \mathfrak{s} & : & \textit{Assgnmnt} \\ \mathfrak{l} & : & \textit{Assgnmnt} \\ \mathfrak{r} & : & \textit{Assgnmnt} \\ \mathfrak{w} & : & \textit{Assgnmnt} \\ \mathfrak{g} & : & \textit{Assgnmnt} \\ \mathfrak{c} & : & \textit{PropCntxt} \end{bmatrix}$

CntxtType — a basic type

T : *CntxtType* iff $T \sqsubseteq$ *Cntxt*

xType — a basic type

T : *xType* iff T : *RecType* and x $\in \text{labels}(T)$

Ppty — $\begin{bmatrix} \text{bg} & : & \textit{xType} \\ \text{fg} & : & (\text{bg}{\rightarrow}\textit{RecType}) \end{bmatrix}$

purification of properties, $\mathcal{P}(P)$

If P : *Ppty*, then

if P.bg^x = P.bg, then

$\mathfrak{P}(P) = P$

otherwise:

$\mathfrak{P}(P)$ is $\ulcorner\lambda r{:}P.\text{bg}^{\text{x}}\ .\ \begin{bmatrix} \mathfrak{c} & : & P.\text{bg} \parallel \left[\text{x=}r.\text{x}\right] \\ \text{e} & : & P(\mathfrak{c}) \end{bmatrix}\urcorner$

purification$^\forall$ of properties, $\mathcal{P}^\forall(P)$

If P : *Ppty*, then

if P.bg^x = P.bg, then

$\mathfrak{P}^\forall(P) = P$

otherwise:

$\mathfrak{P}^\forall(P)$ is $\ulcorner\lambda r{:}P.\text{bg}^{\text{x}}\ .\ ((r'{:}P.\text{bg}\parallel\left[\text{x=}r.\text{x}\right]) \rightarrow \left[\ \text{e} \ : \ P(r')\ \right])\urcorner$

$P\{a\}$

If P is a pure property, $P\{a\}$ represents the type $P(\left[\text{x=}a\right])$

$\mathfrak{T}(P)$

If P : *Ppty* and P is pure, then $\mathfrak{T}(P)$: *Type*.

$a : \mathfrak{T}(P)$ iff $\mathfrak{P}(P)\{a\}$ is witnessed.

exist$^{\text{w}}(P)$

If P : *Ppty*, then exist$^{\text{w}}(P)$: *Type*.

X : exist$^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$

2. $|X| = 1$
(equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) = \frac{1}{|[\check{\mathfrak{T}}(P)]|}$)

exist$^{\mathrm{w}}_{\mathrm{pl}}(P)$

If P : *Ppty*, then exist$^{\mathrm{w}}_{\mathrm{pl}}(P)$: *Type*.

X : exist$^{\mathrm{w}}_{\mathrm{pl}}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $|X| \geq 2$
(equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \frac{2}{|[\check{\mathfrak{T}}(P)]|}$)

no$^{\mathrm{w}}(P)$

If P : *Ppty*, then no$^{\mathrm{w}}(P)$: *Type*.

X : no$^{\mathrm{w}}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $|X| = 0$
(equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) = 0$)

equivalently,

X : no$^{\mathrm{w}}(P)$ iff $X = \emptyset$

every$^{\mathrm{w}}(P)$

If P : *Ppty*, then every$^{\mathrm{w}}(P)$: *Type*.

X : every$^{\mathrm{w}}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $|X| = |[\check{\mathfrak{T}}(P)]|$
(equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) = 1$)

equivalently,

X : every$^{\mathrm{w}}(P)$ iff $X = [\check{\mathfrak{T}}(P)]$

most$^{\mathrm{w}}(P)$

If P : *Ppty*, then most$^{\mathrm{w}}(P)$: *Type*.

X : most$^{\mathrm{w}}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[\downarrow P]|} \geq \theta_{\mathrm{most}}(P)$, where $.5 < \theta_{\mathrm{most}}(P) < 1$
(equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \theta_{\mathrm{most}}(P)$)

many$^{\mathrm{w}}_{a}(P)$

If P : *Ppty*, then many$^{\mathrm{w}}_{a}(P)$: *Type*.

X : many$^{\mathrm{w}}_{a}(P)$ iff

1. $X : \mathrm{set}(\mathfrak{T}(P))$
2. $|X| \geq \theta_{\mathrm{many}_a}(P)$, where $\theta_{\mathrm{many}_a}(P)$ is a natural number, i, such that $i > 2$.
(equivalently, $p(\mathfrak{T}(X)\|\mathfrak{T}(P)) \geq \frac{\theta_{\mathrm{many}_a}(P)}{[[\check{\mathfrak{T}}(P)]]}$)

$\text{many}_p^{\text{w}}(P)$

If P : *Ppty*, then $\text{many}_p^{\text{w}}(P)$: *Type*.

X : $\text{many}_p^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[{\downarrow}P]|} \geq \theta_{\text{many}_p}(P)$, where $0 < \theta_{\text{many}_p}(P) < 1$
   (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \geq \theta_{\text{many}_p}(P)$)

$\text{few}_a^{\text{w}}(P)$

If P : *Ppty*, then $\text{few}_a^{\text{w}}(P)$: *Type*.

X : $\text{few}_a^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \leq \theta_{\text{few}_a}(P)$, where $\theta_{\text{few}_a}(P)$ is a natural number, i, such that $i > 2$
   (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \leq \frac{\theta_{\text{few}_a}(P)}{[[\check{\ }\mathfrak{T}(P)]]}$)

$\text{few}_p^{\text{w}}(P)$

If P : *Ppty*, then $\text{few}_p^{\text{w}}(P)$: *Type*.

X : $\text{few}_p^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[{\downarrow}P]|} \leq \theta_{\text{few}_p}(P)$, where $0 < \theta_{\text{few}_p}(P) < 1$
   (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \leq \theta_{\text{few}_p}(P)$)

$\text{a\_few}_a^{\text{w}}(P)$

If P : *Ppty*, then $\text{a\_few}_a^{\text{w}}(P)$: *Type*.

X : $\text{a\_few}_a^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $|X| \geq \theta_{\text{few}_a}(P)$, where $\theta_{\text{few}_a}(P)$ is a natural number, i, such that $i > 2$
   (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \geq \frac{\theta_{\text{few}_a}(P)}{[[\check{\ }\mathfrak{T}(P)]]}$)

$\text{a\_few}_p^{\text{w}}(P)$

If P : *Ppty*, then $\text{a\_few}_p^{\text{w}}(P)$: *Type*.

X : $\text{a\_few}_p^{\text{w}}(P)$ iff

1. $X : \text{set}(\mathfrak{T}(P))$
2. $\frac{|X|}{|[{\downarrow}P]|} \geq \theta_{\text{few}_p}(P)$, where $0 < \theta_{\text{few}_p}(P) < 1$
   (equivalently, $p(\mathfrak{T}(X) \| \mathfrak{T}(P)) \geq \theta_{\text{few}_p}(P)$)

^{T}*Ppty* — if T is a type, then ^{T}*Ppty* is a type

P : ^{T}*Ppty* iff P : *Ppty* and P.bg $\sqsubseteq \left[\text{x}{:}T\right]$

PlPpty — a basic type

P : *PlPpty* iff P : *Ppty* and for some type T, P.bg $\sqsubseteq \left[\text{x}{:}\text{plurality}(T)\right]$

PPpty — $\left[\begin{array}{lll}\text{bg} & : & \textit{CntxtType}\\ \text{fg} & : & (\text{bg}\rightarrow\textit{Ppty})\end{array}\right]$

$\mathcal{L}(\mathcal{P})$ **New!**

If $\mathcal{P}$ is a parametric property of the form

$\ulcorner\lambda c{:}T_1\,.\ulcorner\lambda r{:}T_2\,.\,\varphi\urcorner\urcorner$

then the *localization of* $\mathcal{P}$, $\mathcal{L}(\mathcal{P})$, is

$\ulcorner\lambda c{:}\textit{Cntxt}\,.\,\ulcorner\lambda r{:}T_2\dot{\wedge}\big[\mathfrak{c}{:}T_1\big]\,.\,\varphi_{c.\pi\rightsquigarrow r.\mathfrak{c}.\pi}\urcorner\urcorner$

$\mathfrak{R}(\mathcal{P})$ **New!**

If $\mathcal{P}$ is a parametric property of the form

$\ulcorner\lambda c{:}T_1\,.\,\ulcorner\lambda r{:}T_2(\!(T_1)\!)\,.\,\varphi(\!(T_1,T_2)\!)\urcorner\urcorner$

where for some natural number i, $T_1\sqsubseteq\big[\mathfrak{r}{:}\big[\text{x}_i{:}\textit{Ind}\big]\big]$, then *the reflexivization of* $\mathcal{P}$, $\mathfrak{R}(\mathcal{P})$, is

$\ulcorner\lambda c{:}(T_1\boxed{\dot{\wedge}}\big[\mathfrak{r}{:}\textit{Assgnmnt}\big])\ominus\mathfrak{s}.\text{x}_i\,.\,\ulcorner\lambda r{:}T_2\,.\,[\varphi]_{c.\mathfrak{s}.\text{x}_i\rightsquigarrow r.\text{x}}\urcorner\urcorner$

^{T}*PPpty* — if T is a type, then ^{T}*PPpty* is a type

$\mathcal{P}$: ^{T}*PPpty* iff $\mathcal{P}$: *PPpty* and for any c : $\mathcal{P}$.bg, $\mathcal{P}(c)$: ^{T}*Ppty*

Quant — (*Ppty*→*RecType*)

PQuant — $\left[\begin{array}{lll}\text{bg} & : & \textit{CntxtType}\\ \text{fg} & : & (\text{bg}\rightarrow\textit{Quant})\end{array}\right]$

QuantDet — (*Ppty*→*Quant*)

PQuantDet — $\left[\begin{array}{lll}\text{bg} & : & \textit{CntxtType}\\ \text{fg} & : & (\text{bg}\rightarrow\textit{QuantDet})\end{array}\right]$

PRecType — $\left[\begin{array}{lll}\text{bg} & : & \textit{CntxtType}\\ \text{fg} & : & (\text{bg}\rightarrow\textit{RecType})\end{array}\right]$

Cont — *PRecType*∨*PPpty*∨*PQuant*∨*PQuantDet*

ContType **New!** — a basic type

T : *ContType* iff $T\sqsubseteq$ *Cont*

$\mathfrak{A}(T)$ **New!**

If T is a parametric content type, then $\mathfrak{A}(T)$ is also a parametric content type.

φ : $\mathfrak{A}(T)$ iff φ : T and φ.bg $\not\sqsubseteq\big[\mathfrak{r}{:}\big[\text{x}_i{:}\textit{Ind}\big]\big]$, for any natural number i.

QStore **New!** — a basic type

r : *QStore* iff r : $\mathit{Assgnmnt}$ and for any $\text{x}_i\in\text{labels}(r)$, $r.\text{x}_i$: *PQuant*

If T is *QStore*$\wedge T'$ and $\pi\in\text{tpaths}(T')$ then

1. if π is ℓ and $\mathrm{labels}(T') = \{\ell\}$, then $T \ominus \pi =$ *PropCntxt*
2. otherwise, $T \ominus \pi =$ *QStore* $\wedge (T' \ominus \pi/T')$

unplugged New! — definition

A parametric content, α, is *unplugged* iff $c : \alpha.\mathrm{bg}$ implies $c.\mathfrak{q} \neq \emptyset$ (that is, $c.\mathfrak{q}$ is not the empty record). Otherwise α is *plugged*.

$\mathrm{store}(\mathcal{Q})$ **New!**

If $\mathcal{Q}$: *PQuant* and $\mathcal{Q}$ is plugged, then $\mathrm{store}(\mathcal{Q})$ is

$$\ulcorner\lambda c{:}\begin{bmatrix} \mathit{Cntxt} \\ \mathfrak{q}{:}\big[\mathrm{x}_0{=}\mathcal{Q}{:}\mathit{PQuant}\big] \\ \mathfrak{s}{:}\big[\mathrm{x}_0{:}\mathit{Ind}\big] \end{bmatrix} \,.\, \lambda P{:}\mathit{Ppty} \,.\, P\{c.\mathfrak{s}.\mathrm{x}_0\}\urcorner$$

$\mathrm{retrieve}(\mathrm{x}_i, \alpha)$ **New!**

If α : *PRecType*, $\mathcal{Q}$: *PQuant* $\alpha.\mathrm{bg} \sqsubseteq \big[\mathfrak{q}{:}\big[\mathrm{x}_i{=}\mathcal{Q}{:}\mathit{Ind}\big]\big]$, $\mathcal{Q}'$ is $[\mathcal{Q}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\mathfrak{f}}$ and α' is $[\mathrm{incr}(\alpha, \mathcal{Q}')]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.a}$, then $\mathrm{retrieve}(\mathrm{x}_i, \alpha)$ is

$$\begin{aligned}&\lambda c{:}(\mathcal{Q}'.\mathrm{bg}\underset{\cdot}{\wedge}\alpha'.\mathrm{bg} \ominus \mathfrak{q}.\mathrm{x}_i, \mathfrak{s}.\mathrm{x}_i) \,. \\ &\quad \mathcal{Q}'(c)(\mathfrak{P}(\ulcorner\lambda r{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind} \\ \mathfrak{s}{:}\begin{bmatrix}\mathit{Assgnmnt} \\ \mathrm{x}_i{=}\Uparrow\mathrm{x}{:}\mathit{Ind}\end{bmatrix}\end{bmatrix} \underset{\cdot}{\wedge}\alpha'.\mathrm{bg}^{\mathfrak{s}.\mathrm{x}_i} \,.\, \alpha'(c[r][\mathfrak{q}.\mathrm{x}_i = \mathcal{Q}])\urcorner))\end{aligned}$$

$\mathfrak{S}(T)$ **New!** — if T : *ContType*, then $\mathfrak{S}(T)$ is a type

The witnesses of $\mathfrak{S}(T)$ are characterized by

1. if $\varphi : T$ then $\varphi : \mathfrak{S}(T)$
2. if $\varphi : \mathfrak{S}(T)$ and φ : *PPpty*, then $\mathcal{L}(\varphi) : \mathfrak{S}(T)$
3. if $\varphi : \mathfrak{S}(T)$ and φ : *PPpty*, then $\mathfrak{R}(\varphi) : \mathfrak{S}(T)$
4. if $\varphi : \mathfrak{S}(T)$, $\varphi \sqsubseteq \big[\mathrm{scope}{=}\psi{:}\mathit{Ppty}\big]$ and $\pi_1, \pi_2 \in \mathrm{paths}(\psi.\mathrm{bg})$, then $\varphi[\mathrm{scope} = \psi_{\pi_1=\pi_2} : \mathit{Ppty}] : \mathfrak{S}(T)$
5. if $\alpha\mathcal{O}\beta : \mathfrak{S}(T)$, (for some combination operation, $\mathcal{O}$) and $\alpha\mathcal{O}_{i,j}\beta$ is defined (for some natural numbers, i and j), then $\alpha\mathcal{O}_{i,j}\beta : \mathfrak{S}(T)$
6. if $\varphi : \mathfrak{S}(T)$ and φ is in the range of 'store', then $\mathrm{store}(\varphi) : \mathfrak{S}(T)$
7. if $\varphi : \mathfrak{S}(T)$ and 'x_i' and φ are appropriate arguments to 'retrieve', then $\mathrm{retrieve}(\mathrm{x}_i, \varphi) : \mathfrak{S}(T)$
8. nothing is a witness for $\mathfrak{S}(T)$ except as required above.

Cat — a basic type

s, np, det, n, v, vp : *Cat*

Syn — $\begin{bmatrix} \mathrm{cat} & : & \mathit{Cat} \\ \mathrm{daughters} & : & \mathit{Sign}^* \end{bmatrix}$

Sign **Revised!** — a basic type

$$\sigma : \textit{Sign} \text{ iff } \sigma : \begin{bmatrix} \text{s-event} & : & \textit{SEvent} \\ \text{syn} & : & \textit{Syn} \\ \text{cont} & : & \textit{ContType} \end{bmatrix}$$

SignType — a basic type

T : *SignType* iff $T \sqsubseteq$ *Sign*

S — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=s:}\textit{Cat}\end{bmatrix} \end{bmatrix}$)

S/i — if i is a natural number, then *S/i* is a type

α : *S*/i iff α : *S* and α.cont.bg $\sqsubseteq \begin{bmatrix}\mathfrak{g}\text{:}\begin{bmatrix}\text{x}_i\text{:}\textit{Ind}\end{bmatrix}\end{bmatrix}$

NP — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=np:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

whNP — a basic type

σ : *WhNP* iff σ : *NP*, σ.cont is $\mathcal{Q}$ and $\mathcal{Q}$.bg $\sqsubseteq \begin{bmatrix}\mathfrak{w}\text{:}\begin{bmatrix}\text{x}_i\text{:}\textit{Ind}\end{bmatrix}\end{bmatrix}$, for some natural number i.

$\textit{NP}_{\text{wh}_i}$ — if i is a natural number, then $\textit{NP}_{\text{wh}_i}$ is a type

α : $\textit{NP}_{\text{wh}_i}$ iff α : *NP* and α.cont.bg $\sqsubseteq \begin{bmatrix}\mathfrak{w}\text{:}\begin{bmatrix}\text{x}_i\text{:}\textit{Ind}\end{bmatrix}\end{bmatrix}$

Det — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=det:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

N — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=n:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

${}^{T}\!N$ — if T is a type, then ${}^{T}\!N$ is a type

α : ${}^{T}\!N$ iff α : *N* and α.cont : ${}^{T}\textit{PPpty}$

V — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=v:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

VP — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=vp:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

Rel — $\begin{bmatrix} \textit{Sign} \\ \text{syn:}\begin{bmatrix}\text{cat=rel:}\textit{Cat}\end{bmatrix} \end{bmatrix}$

${}^{T}\textit{Rel}$ — if T is a type, then ${}^{T}\textit{Rel}$ is a type

α : ${}^{T}\textit{Rel}$ iff α : *Rel* and α.cont : ${}^{T}\textit{PPpty}$

NoDaughters — $\begin{bmatrix}\text{syn:}\begin{bmatrix}\text{daughters=}\varepsilon\text{:}\textit{Sign}^{*}\end{bmatrix}\end{bmatrix}$

Real — a basic type

n : *Real* iff n is a real number

Card — a basic type

n : *Card* iff n is a cardinal number (natural numbers with the addition of $\aleph_0, \aleph_1, \ldots$)

$$\mathit{AmbTempFrame} \text{ — } \begin{bmatrix} \text{x} & : & \mathit{Real} \\ \text{loc} & : & \mathit{Loc} \\ \text{e} & : & \text{temp(loc, x)} \end{bmatrix}$$

$$\mathit{TempRiseEventCntxt} \text{ — } \begin{bmatrix} \text{fix} & : & \begin{bmatrix} \text{loc} & : & \mathit{Loc} \end{bmatrix} \\ \text{scale} & : & (\mathit{AmbTempFrame} \to \mathit{Real}) \end{bmatrix}$$

TempRiseEvent —

$$\lambda r{:}\mathit{TempRiseEventCntxt}\ .\ \begin{bmatrix} \text{e} & : & (\mathit{AmbTempFrame} \| r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1]) \end{bmatrix}$$

$$\mathit{PriceFrame} \text{ — } \begin{bmatrix} \text{x} & : & \mathit{Real} \\ \text{loc} & : & \mathit{Loc} \\ \text{commodity} & : & \mathit{Ind} \\ \text{e} & : & \text{price(commodity, loc, x)} \end{bmatrix}$$

$$\mathit{PriceRiseEventCntxt} \text{ — } \begin{bmatrix} \text{fix} & : & \begin{bmatrix} \text{loc} & : & \mathit{Loc} \\ \text{commodity} & : & \mathit{Ind} \end{bmatrix} \\ \text{scale} & : & (\mathit{PriceFrame} \to \mathit{Real}) \end{bmatrix}$$

PriceRiseEvent —

$$\lambda r{:}\mathit{TempRiseEventCntxt}\ .\ \begin{bmatrix} \text{e} & : & (\mathit{PriceFrame} \| r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1]) \end{bmatrix}$$

$$\mathit{LocFrame} \text{ — } \begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{loc} & : & \mathit{Loc} \\ \text{e} & : & \text{at(x, loc)} \end{bmatrix}$$

$$\mathit{LocRiseEventCntxt} \text{ — } \begin{bmatrix} \text{fix} & : & \begin{bmatrix} \text{x} & : & \mathit{Ind} \end{bmatrix} \\ \text{scale} & : & (\mathit{LocFrame} \to \mathit{Real}) \end{bmatrix}$$

LocRiseEvent —

$$\lambda r{:}\mathit{LocRiseEventCntxt}\ .\ \begin{bmatrix} \text{e} & : & (\mathit{LocFrame} \| r.\text{fix})^2 \\ \text{c}_{\text{rise}} & : & r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1]) \end{bmatrix}$$

Topos — a basic type

$$\text{If } \tau : \mathit{Topos}\text{, then } \tau : \begin{bmatrix} \text{bg} & : & \mathit{Type} \\ \text{fg} & : & (\text{bg} \to \mathit{Type}) \end{bmatrix}$$

8.4.1.2 Predicates

(as in Chapter 7)

8.4.1.3 Properties

(as in Chapter 7)

8.4.1.4 Scales

(as in Chapter 5)

8.4.1.5 Lexicon

Lex

If T_{phon} is a phonological type (that is, $T_{\text{phon}} \sqsubseteq \textit{Phon}$) and T_{sign} is a sign type (that is, $T_{\text{sign}} \sqsubseteq \textit{Sign}$), then we shall use Lex(T_{phon}, T_{sign}) to represent

$$((T_{\text{sign}} \mathbin{\dot\wedge} \left[\text{s-event:}\left[\text{e:}T_{\text{phon}}\right]\right]) \mathbin{\dot\wedge} \textit{NoDaughters})$$

SemCommonNoun(T_{bg}, p)

If p is a predicate with arity $\langle \textit{Ind} \rangle$ and T_{bg} is a type (of context), then SemCommonNoun(T_{bg}, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\left[\text{x:}\textit{Ind}\right]\ .\ \left[\ \text{e} \quad : \quad p(r.\text{x})\ \right]\urcorner\urcorner$$

If p is a predicate with arity $\langle \textit{Rec} \rangle$ and T_{bg} is a type (of context), then SemCommonNoun(T_{bg}, p) is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\left[\text{x:}\textit{Rec}\right]\ .\ \left[\ \text{e} \quad : \quad p(r.\text{x})\ \right]\urcorner\urcorner$$

$\text{Lex}_{\text{CommonNoun}}$($T_{\text{phon}}$, T_{bg}, p) **Revised!**

If T_{phon} is a phonological type, p is a predicate with arity $\langle \textit{Ind} \rangle$ or $\langle \textit{Rec} \rangle$ and T_{bg} is a type (of context), then $\text{Lex}_{\text{CommonNoun}}$($T_{\text{phon}}$, T_{bg}, p) is

$$\text{Lex}(T_{\text{phon}}, N) \mathbin{\dot\wedge} \left[\text{cont=SemCommonNoun}(T_{\text{bg}}, p)^{\mathfrak{S}}{:}\textit{ContType}\right]$$

SemPropName(T_{phon})

If T_{phon} is a phonological type, then SemPropName(T_{phon}) is

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix}\text{x:}\textit{Ind} \\ \text{e:named(x, }T_{\text{phon}}\text{)}\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(c.\mathfrak{c})\urcorner$$

$\text{Lex}_{\text{PropName}}(T_{\text{phon}})$ **Revised!**

If T_{phon} is a phonological type,

then $\text{Lex}_{\text{PropName}}(T_{\text{phon}})$ is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot{\wedge}} \left[\text{cnt}{=}\text{SemPropName}(T_{\text{phon}})^{\mathfrak{S}}{:}\textit{ContType}\right]$$

SemPron **Revised!**

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}{:}\left[\text{x}_0{:}\textit{Ind}\right]\\ \mathfrak{l}{:}\left[\text{x}_0{:}\textit{Ind}\right]\end{bmatrix} .\ \lambda P{:}\textit{Ppty}\ .\ P(\left[\text{x}{=}c.\mathfrak{s}.\text{x}_0\right])\urcorner$$

LexPron(T_{phon}) **Revised!**

If T_{phon} is a phonological type, then LexPron(T_{phon}) is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot{\wedge}} \left[\text{cont}{=}\text{SemPron}^{\mathfrak{S}}{:}\textit{ContType}\right]$$

SemWhPron

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{w}{:}\left[\text{x}_0{:}\textit{Ind}\right]\end{bmatrix} .\ \lambda P{:}\textit{Ppty}\ .\ P(\left[\text{x}{=}c.\mathfrak{w}.\text{x}_0\right])\urcorner$$

LexWhPron(T_{phon}) **Revised!**

If T_{phon} is a phonological type, then LexWhPron(T_{phon}) is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot{\wedge}} \left[\text{cont}{=}\text{SemWhPron}^{\mathfrak{S}}{:}\textit{ContType}\right]$$

SemReflPron **New!**

$$\ulcorner\lambda c{:}\begin{bmatrix}\textit{Cntxt}\\ \mathfrak{s}{:}\left[\text{x}_0{:}\textit{Ind}\right]\\ \mathfrak{r}{:}\left[\text{x}_0{:}\textit{Ind}\right]\end{bmatrix} .\ \lambda P{:}\textit{Ppty}\ .\ P\{c.\mathfrak{s}.\text{x}_0\}\urcorner$$

LexReflPron(T_{phon}) **New!**

If T_{phon} is a phonological type, then LexReflPron(T_{phon}) is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot{\wedge}} \left[\text{cont}{=}\text{SemReflPron}^{\mathfrak{S}}{:}\textit{ContType}\right]$$

SemNumeral(n)

If n is a real number, then SemNumeral(n) is

$$\ulcorner\lambda c{:}\textit{Cntxt}\ .\ \lambda P{:}\textit{Ppty}\ .\ P(\left[\text{x}{=}n\right])\urcorner$$

$\text{Lex}_{\text{numeral}}(T_{\text{phon}}, n)$ **Revised!**

If T_{phon} is a phonological type and n is a real number, then $\text{Lex}_{\text{numeral}}(T_{\text{phon}}, n)$ is

$$\text{Lex}(T_{\text{phon}}, \textit{NP}) \mathbin{\dot{\wedge}} \left[\text{cnt}{=}\text{SemNumeral}(n)^{\mathfrak{S}}{:}\textit{ContType}\right]$$

SemIndefArt

$$\lambda Q{:}\mathit{Ppty}\ .\ \ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix}\text{restr}{=}Q & : & \mathit{Ppty}\\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty}\\ \text{e} & : & \text{exist(restr, scope)}\end{bmatrix}\urcorner$$

$\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ **Revised!**

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{Det}) \mathbin{\dot\wedge} \begin{bmatrix}\text{cont}{=}\text{SemIndefArt}^{\mathfrak{S}}{:}\mathit{ContType}\end{bmatrix}$$

SemUniversal

$$\lambda Q{:}\mathit{Ppty}\ .\ \ulcorner\lambda c{:}\mathit{Cntxt}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix}\text{restr}{=}Q & : & \mathit{Ppty}\\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty}\\ \text{e} & : & \text{every(restr, scope)}\end{bmatrix}\urcorner$$

$\text{Lex}_{\text{Universal}}(T_{\text{Phon}})$ **Revised!**

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{Universal}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{Det}) \mathbin{\dot\wedge} \begin{bmatrix}\text{cont}{=}\text{SemUniversal}^{\mathfrak{S}}{:}\mathit{ContType}\end{bmatrix}$$

SemDefArt

$$\lambda Q{:}\mathit{Ppty}\ .\ \ulcorner\lambda c{:}\begin{bmatrix}\mathit{Cntxt}\\ \mathfrak{c}{:}\begin{bmatrix}\text{e}{:}\text{unique}(Q)\end{bmatrix}\end{bmatrix}\ .\ \lambda P{:}\mathit{Ppty}\ .\ \begin{bmatrix}\text{restr}{=}Q\restriction c.\mathfrak{c}.\text{e} & : & \mathit{Ppty}\\ \text{scope}{=}P|_{\mathfrak{F}(\text{restr})} & : & \mathit{Ppty}\\ \text{e} & : & \text{every(restr, scope)}\end{bmatrix}\urcorner$$

$\text{Lex}_{\text{DefArt}}(T_{\text{Phon}})$ **Revised!**

If T_{Phon} is a phonological type, then $\text{Lex}_{\text{IndefArt}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, \mathit{Det}) \mathbin{\dot\wedge} \begin{bmatrix}\text{cont}{=}\text{SemDefArt}^{\mathfrak{S}}{:}\mathit{ContType}\end{bmatrix}$$

$\text{SemIntransVerb}(T_{\text{bg}}, p)$

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \mathit{Ind}\rangle$, then $\text{SemIntransVerb}(T_{\text{bg}}, p)$ is

$$\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\lambda r{:}\begin{bmatrix}\text{x}{:}\mathit{Ind}\end{bmatrix}\ .\ \begin{bmatrix}\text{e} & : & p(r.\text{x})\end{bmatrix}\urcorner\urcorner$$

If $T_{\text{bg}} \sqsubseteq \left[\mathfrak{c}{:}\textit{Rec}\right]$ is a record type (for context) and p is a predicate with arity $\langle \textit{Rec}, \textit{Rec} \rangle$, then SemIntransVerb($T_{\text{bg}}$, p) is

$$\ulcorner \lambda c{:}T_{\text{bg}} \,.\, \ulcorner \lambda r{:}\left[\text{x}{:}\textit{Rec}\right] \,.\, \left[\ \text{e} \ :\ p(r.\text{x}, c.\mathfrak{c})\ \right] \urcorner \urcorner$$

Lex$_{\text{IntransVerb}}$(T_{phon}, T_{bg}, p) **Revised!**

If T_{phon} is a phonological type, $T_{\text{bg}} \sqsubseteq \left[\mathfrak{c}{:}\textit{Rec}\right]$ a record type (for context) and p is a predicate with arity $\langle \textit{Ind} \rangle$ or $\langle \textit{Rec}, \textit{Rec} \rangle$, then Lex$_{\text{IntransVerb}}$($T_{\text{phon}}$, T_{bg}, p) is

$$\text{Lex}(T_{\text{phon}}, V_i) \mathbin{\dot{\wedge}} \left[\text{cnt=SemIntransVerb}(T_{\text{bg}}, p)^{\mathfrak{S}}{:}\textit{ContType}\right]$$

SemTransVerb(T_{bg}, p)

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \textit{Ind}, \textit{Ind} \rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner \lambda c{:}T_{\text{bg}} \,.\, \lambda \mathcal{Q}{:}\textit{Quant} \,.\, \ulcorner \lambda r_1{:}\left[\text{x}{:}\textit{Ind}\right] \,.\, \mathcal{Q}(\ulcorner \lambda r_2{:}\left[\text{x}{:}\textit{Ind}\right] \,.\, \left[\ \text{e} \ :\ p(r_1.\text{x}, r_2.\text{x})\ \right] \urcorner) \urcorner \urcorner$$

If T_{bg} is a record type (for context) and p is a predicate with arity $\langle \textit{Ind}, \textit{Quant} \rangle$, then SemTransVerb($T_{\text{bg}}$, p) is

$$\ulcorner \lambda c{:}T_{\text{bg}} \,.\, \lambda \mathcal{Q}{:}\textit{Quant} \,.\, \ulcorner \lambda r{:}\left[\text{x}{:}\textit{Ind}\right] \,.\, \left[\ \text{e} \ :\ p(r.\text{x}, \mathcal{Q})\ \right] \urcorner) \urcorner$$

Lex$_{\text{TransVerb}}$(T_{phon}, T_{bg}, p) **Revised!**

If T_{phon} is a phonological type, T_{bg} a record type (for context) and p is a predicate with arity $\langle \textit{Ind}, \textit{Ind} \rangle$ or $\langle \textit{Ind}, \textit{Quant} \rangle$, then Lex$_{\text{TransVerb}}$($T_{\text{phon}}$, T_{bg}, p) is

$$\text{Lex}(T_{\text{phon}}, V_t) \mathbin{\dot{\wedge}} \left[\text{cnt=SemTransVerb}(T_{\text{bg}}, p)^{\mathfrak{S}}{:}\textit{ContType}\right]$$

SemBe

SemBe$_{\text{ID}}$

$$\begin{array}{l}
\ulcorner \lambda c{:}\begin{bmatrix} \textit{Cntxt} \\ \mathfrak{c}{:}\left[\text{ty}{:}\textit{Type}\right] \end{bmatrix} . \\
\quad \lambda \mathcal{Q}{:}\textit{Quant} \,. \\
\qquad \ulcorner \lambda r_1{:}\left[\text{x}{:}c.\mathfrak{c}.\text{ty}\right] . \\
\qquad\quad \mathcal{Q}(\ulcorner \lambda r_2{:}\left[\text{x}{:}c.\mathfrak{c}.\text{ty}\right] . \begin{bmatrix} \text{x=}r_1.\text{x},\ r_2.\text{x} & : & c.\mathfrak{c}.\text{ty} \\ \text{e} & : & \text{be(x)} \end{bmatrix} \urcorner) \urcorner \urcorner
\end{array}$$

SemBe$_{\text{scalar}}$

$$\begin{array}{l}
\ulcorner \lambda c{:}\begin{bmatrix} \textit{Cntxt} \\ \mathfrak{c}{:}\begin{bmatrix} \text{ty}{:}\textit{Type} \\ \text{sc}{:}(\text{ty}\rightarrow\textit{Real}) \end{bmatrix} \end{bmatrix} . \\
\quad \lambda \mathcal{Q}{:}\textit{Quant} \,. \\
\qquad \ulcorner \lambda r_1{:}\left[\text{x}{:}c.\mathfrak{c}.\text{ty}\right] . \\
\qquad\quad \mathcal{Q}(\ulcorner \lambda r_2{:}\left[\text{x}{:}\textit{Real}\right] . \begin{bmatrix} \text{x=}c.\mathfrak{c}.\text{sc}(r_1.\text{x}),\ r_2.\text{x} & : & \textit{Real} \\ \text{e} & : & \text{be(x)} \end{bmatrix} \urcorner) \urcorner \urcorner
\end{array}$$

Lex$_{\text{be}}(T_{\text{Phon}})$ **Revised!**

If T_{Phon} is a phonological type, then Lex$_{\text{be}_{\text{ID}}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, V) \mathbin{\dot\wedge} \left[\text{cont=SemBe}_{\text{ID}}{}^{\mathfrak{S}}\text{:}\mathit{ContType}\right]$$

If T_{Phon} is a phonological type, then Lex$_{\text{be}_{\text{scalar}}}(T_{\text{Phon}})$ is

$$\text{Lex}(T_{\text{Phon}}, V) \mathbin{\dot\wedge} \left[\text{cont=SemBe}_{\text{scalar}}{}^{\mathfrak{S}}\text{:}\mathit{ContType}\right]$$

$\text{FrameType}(p)$

FrameType is a partial function on predicates, p, with arity $\langle \mathit{Ind} \rangle$ which can be defined for particular agents and particular times, which obeys the constraint:

$$\text{FrameType}(p) \sqsubseteq \begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{e} & : & p(\text{x}) \end{bmatrix}$$

p_frame

1. If p is a predicate in the domain of FrameType, then p_frame is a predicate with arity $\langle \mathit{Rec} \rangle$.
2. e : p_frame(r) iff r : $\text{FrameType}(p)$ and $e = r$

p_pl

1. If p is a singular predicate (i.e. there is no p' such that $p = p'$_pl) with arity $\langle T \rangle$, then p_pl is a predicate with arity $\langle \text{plurality}(T) \rangle$
2. e : p_pl(A) if for all $a \in A$, $e : p(a)$

CommonNounIndToFrame

If T_{phon} is a phonological type, p is a predicate with arity $\langle \mathit{Ind} \rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then
CommonNounIndToFrame(Lex$_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$$\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p\text{\_frame})$$

RestrictCommonNoun **Revised!**

If T_{phon} is a phonological type, p is a predicate, T_{bg} and T_{res} are record types and Σ is Lex$_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p)$, then RestrictCommonNoun(Σ, T_{res}) is

$$\Sigma \boxed{\dot\wedge} \left[\text{cont=}\ulcorner\lambda c{:}T_{\text{bg}}\ .\ \ulcorner\text{SemCommonNoun}(T_{\text{bg}}, p)(c)\,|_{T_{\text{res}}}\urcorner\urcorner^{\mathfrak{S}} \quad : \quad \mathit{ContType} \right]$$

IntransVerbIndToFrame

If T_{phon} is a phonological type, p is a predicate with arity $\langle \mathit{Ind} \rangle$ and T_{bg} is a record type (the "background type" or "presupposition") then
IntransVerbIndToFrame(Lex$_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$) =

$\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p\_\text{frame})$

PluralCommonNoun

We assume that 'pluralnoun' is a function that maps phonological types for singular common nouns to corresponding phonological types for plural common nouns.

If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T\rangle$ and T_{bg} is a record type then $\text{PluralCommonNoun}(\text{Lex}_{\text{CommonNoun}}(T_{\text{phon}}, T_{\text{bg}}, p))$ =

$$\text{Lex}_{\text{CommonNoun}}(\text{pluralnoun}(T_{\text{phon}}), T_{\text{bg}}, p\_\text{pl})$$

PluralIntransVerb

We assume that 'pluralverb' is a function that maps phonological types for singular verbs to corresponding phonological types for plural verbs.

If T_{phon} is a (singular) phonological type, p is a singular predicate with arity $\langle T\rangle$ and T_{bg} is a record type then $\text{PluralIntransVerb}(\text{Lex}_{\text{IntransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p))$ =

$$\text{Lex}_{\text{IntransVerb}}(\text{pluralverb}(T_{\text{phon}}), T_{\text{bg}}, p\_\text{pl})$$

TransVerbToVerbPhrase **Revised!**

If T_{phon} is a phonological type, T_{bg} a context type, p is a predicate with arity $\langle \mathit{Ind}, \mathit{Ind}\rangle$ or $\langle \mathit{Ind}, \mathit{Quant}\rangle$ and Σ is $\text{Lex}_{\text{TransVerb}}(T_{\text{phon}}, T_{\text{bg}}, p)$, then $\text{TransVerbToVerbPhrase}(\Sigma)$ is

$$\Sigma \boxed{\underset{\cdot}{\wedge}} \begin{bmatrix} \text{cat=vp} & : & \mathit{Cat} \\ \text{cont=}\Sigma\text{.cont}/\mathit{Quant} & : & \mathit{ContType} \end{bmatrix}$$

where for any content type, T, such that $\varphi : T$ implies $\varphi : (\mathit{Quant} \rightarrow \mathit{PPpty})$, T/Quant is a type such that

$$\varphi : T \text{ iff } \ulcorner\lambda c{:}T.\text{bg}\underset{\cdot}{\wedge}\big[\mathfrak{g}{:}\big[\text{x}_0{:}\mathit{Ind}\big]\big] \;.\; \varphi(c)(\lambda P{:}\mathit{Ppty} \;.\; P\{c.\mathfrak{g}.\text{x}_0\})\urcorner : T/\mathit{Quant}$$

8.4.1.6 Constituent structure

$\text{RuleDaughters}(T_{\text{daughters}}, T_{\text{mother}})$

If T_{mother} is a sign type and $T_{\text{daughters}}$ is a type of strings of signs then

$$\text{RuleDaughters}(T_{\text{daughters}}, T_{\text{mother}})$$

is

$$\lambda u{:}T_{\text{daughters}} \;.\; T_{\text{mother}} \underset{\cdot}{\wedge} \big[\text{syn:}\big[\text{daughters=}u{:}T_{\text{daughters}}\big]\big]$$

ConcatPhon

$$\lambda u{:}\big[\text{s-event:}\big[\text{e:}\mathit{Phon}\big]\big]^{+} \;.$$
$$\big[\ \text{s-event} \ : \ \big[\ \text{e=concat}_i(u[i].\text{s-event.e}) \ : \ \mathit{Phon}\ \big]\ \big]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$

If T_{mother} is a sign type and $T_{\text{daughter}_1}, \ldots T_{\text{daughter}_n}$ are sign types, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1} \ldots T_{\text{daughter}_n}$$

represents

$$\text{RuleDaughters}(T_{\text{mother}}, T_{\text{daughter}_1}{}^\frown \ldots {}^\frown T_{\text{daughter}_n}) \dot{\wedge} \text{ConcatPhon}$$

$B(\alpha)$ **New!**

If α is a parametric content,

$$\ulcorner \lambda c{:}T \,.\, \varphi(\!(c)\!)\urcorner$$

then $B(\alpha)$ is

$$\ulcorner \lambda c{:}T \boxed{\dot{\wedge}} \left[\mathfrak{l}{:}\mathit{Assgnmnt}\right] \,.\, \varphi(\!(c)\!)\urcorner$$

$\alpha@\beta$

If $\alpha : \begin{bmatrix}\text{bg}{:}\mathit{CntxtType} \\ \text{fg}{:}(\text{bg}\rightarrow(T_1 \rightarrow T_2))\end{bmatrix}$ and $\beta : \begin{bmatrix}\text{bg}{:}\mathit{CntxtType} \\ \text{fg}{:}(\text{bg}\rightarrow T_1)\end{bmatrix}$ then the *combination of* α *and* β *based on functional application*, $\alpha@\beta$, is

$$\ulcorner \lambda c{:}[\alpha.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}} \dot{\wedge} \text{incr}([\beta.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}, \alpha.\text{bg}) \,. \\ [\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}(c)(\text{incr}([\beta.\text{fg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}, \alpha.\text{bg})(c))\urcorner$$

$\alpha@_{\text{wh}_{i,j}}\beta$

If

1. $\alpha : \begin{bmatrix}\text{bg}{:}\mathit{CntxtType} \\ \text{fg}{:}(\text{bg}\rightarrow\mathit{Quant})\end{bmatrix}$,
2. $\beta : \begin{bmatrix}\text{bg}{:}\mathit{CntxtType} \\ \text{fg}{:}(\text{bg}\rightarrow\mathit{RecType})\end{bmatrix}$,
3. $\alpha.\text{bg} \sqsubseteq \left[\mathfrak{w}{:}\left[\text{x}_i{:}\mathit{Ind}\right]\right]$ for some natural number, i, and
4. $\beta.\text{bg} \sqsubseteq \left[\mathfrak{g}{:}\left[\text{x}_j{:}\mathit{Ind}\right]\right]$ for some natural number, j,

then *the wh*$_{i,j}$*-combination of* α *and* β, $\alpha@_{\text{wh}_{i,j}}\beta$, is

$$\ulcorner \lambda c{:}([\alpha.\text{bg}\ominus\text{paths}_{\mathfrak{w}.\text{x}_i}(\alpha.\text{bg})]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}\dot{\wedge} \\ \text{incr}([\beta.\text{bg} \ominus \text{paths}_{\mathfrak{g}.\text{x}_j}(\beta.\text{bg})]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.a}, \alpha.\text{bg})) \,. \\ \mathfrak{P}(\ulcorner \lambda r_1{:}[\alpha.\text{bg}^{\mathfrak{w}.\text{x}_i}]_{\mathfrak{w}.\text{x}_i\rightsquigarrow\text{x}} \,. \\ \alpha_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f},\mathfrak{w}.\text{x}_i\rightsquigarrow\text{x}}(c[r_1])\ (\mathfrak{P}(\ulcorner \lambda r_2{:}[\beta.\text{bg}^{\mathfrak{g}.\text{x}_j}]_{\mathfrak{g}.\text{x}_j\rightsquigarrow\text{x}} \,. \\ \text{incr}(\beta_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a},\mathfrak{g}.\text{x}_j\rightsquigarrow\text{x}}, \alpha.\text{bg})(c[r_2])\urcorner))\urcorner)\urcorner$$

$\alpha@@\beta$

If $\alpha : (T_1 \rightarrow \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow T_2)\end{bmatrix})$ and $\beta : \begin{bmatrix}\text{bg:}\mathit{CntxtType}\\ \text{fg:(bg}\rightarrow T_1)\end{bmatrix}$ then the *combination of* α *and* β *based on functional application*, $\alpha@@\beta$, is

$$\ulcorner\lambda c{:}\begin{bmatrix}\mathfrak{c} & : & \begin{bmatrix}\text{s} & : & \beta.\text{bg}\\ \text{f} & : & \alpha(\beta(s)).\text{bg}\\ \text{a=s.}\mathfrak{c} & : & \mathit{PropCntxt}\end{bmatrix}\\ \mathfrak{s}\text{=}\mathfrak{c}\text{.s.}\mathfrak{s} & : & \mathit{Assgnmnt}\end{bmatrix}\ .$$

$$[\alpha]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}([\beta]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}(c))(c)\urcorner$$

$\alpha@_{\&}\beta$

If T is a type, $\alpha : {}^{T}\mathit{PPpty}$ and $\beta : {}^{T}\mathit{PPpty}$ then *the property conjunction combination of* α *and* β, $\alpha@_{\&}\beta$, is

$$\lambda c{:}[\alpha.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}\dot{\wedge}\text{incr}([\beta.\text{bg}]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}, \alpha.\text{bg})\ .\ \alpha_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{f}}(c)\&\text{incr}([\beta]_{\mathfrak{c}\rightsquigarrow\mathfrak{c}.\text{a}}, \alpha.\text{bg})(c)$$

$\alpha\mathcal{O}_{i,j}\beta$ **New!**

If $\mathcal{O}$ is a combination operator, then so is $\mathcal{O}_{i,j}$, where i and j are natural numbers.

If α and β are parametric contents such that

1. $\alpha\mathcal{O}\beta$ is defined
2. $\alpha.\text{bg} \sqsubseteq \left[\mathfrak{s}{:}\left[\text{x}_i{:}\mathit{Ind}\right]\right]$
3. $\text{incr}(\beta.\text{bg}, \alpha.\text{bg}) \sqsubseteq \left[\mathfrak{s}{:}\left[\text{x}_j{:}\mathit{Ind}\right]\right]$
4. $\text{incr}(\beta.\text{bg}, \alpha.\text{bg}) \not\sqsubseteq \left[\mathfrak{q}{:}\left[\text{x}_j{:}\mathit{PQuant}\right]\right]$
5. $\text{incr}(\beta.\text{bg}, \alpha.\text{bg}) \not\sqsubseteq \left[\mathfrak{l}{:}\left[\text{x}_j{:}\mathit{PQuant}\right]\right]$

then $\alpha\mathcal{O}_{i,j}\beta$ is

$$[\alpha\mathcal{O}\beta]_{\mathfrak{s}.\text{x}_j\rightsquigarrow\mathfrak{s}.\text{x}_i}$$

$T_1\mathcal{O}^{\mathfrak{S}}T_2$ **New!**

Suppose that T_1 and T_2 are of type *ContType* and that $\mathcal{O}$ is a combination operation such as @,... as characterized above, then we say $T_1\mathcal{O}^{\mathfrak{S}}T_2$ is also a type with the witness condition:

> If $\alpha : T_1$, $\beta : T_2$ and $\alpha\mathcal{O}\beta$ is defined, then $\alpha\mathcal{O}\beta : T_1\mathcal{O}^{\mathfrak{S}}T_2$. Nothing else is a witness for $T_1\mathcal{O}^{\mathfrak{S}}T_2$.

$T_1\mathcal{O}^{\mathfrak{S},B}T_2$ **New!**

Suppose that T_1 and T_2 are of type *ContType* and that $\mathcal{O}$ is a combination operation such as @ etc., then we say that $T_1\mathcal{O}^{\mathfrak{S},B}T_2$ is also a type with the witness condition:

If $\alpha : T_1$, $\beta : T_2$ and $\alpha\mathcal{O}\beta$ is defined, then $B(\alpha\mathcal{O}\beta) : T_1\mathcal{O}^{\mathfrak{S},B}T_2$. Nothing else is a witness for $T_1\mathcal{O}^{\mathfrak{S},B}T_2$.

ContForwardApp$_{\mathfrak{S},\mathcal{O}}$ **New!**

If $\mathcal{O}$ is one of @,... as defined above, then ContForwardApp$_{\mathfrak{S},\mathcal{O}}$ is

$$\lambda u{:}\big[\text{cont:}\textit{ContType}\big]^\frown\big[\text{cont:}\textit{ContType}\big]\ .\ \big[\text{cont:}\mathfrak{S}(u[0].\text{cont}\mathcal{O}^{\mathfrak{S}}u[1].\text{cont})\big]$$

Note: This single characterization of ContForwardApp replaces all the previous variants that were defined in previous chapters.

ContForwardApp$_{\mathfrak{S},\mathcal{O},B}$ **New!**

If $\mathcal{O}$ is a combination operator, then ContForwardApp$_{\mathfrak{S},\mathcal{O},B}$ is

$$\lambda u{:}\big[\text{cont:}\textit{ContType}\big]^\frown\big[\text{cont:}\textit{ContType}\big]\ .\ \big[\text{cont:}\mathfrak{S}(u[0].\text{cont}\mathcal{O}^{\mathfrak{S},B}u[1].\text{cont})\big]$$

ContForwardApp$_{\mathfrak{S},\mathcal{O},\mathfrak{A}}$ **New!**

If $\mathcal{O}$ is a combination operator, then ContForwardApp$_{\mathfrak{S},\mathcal{O},\mathfrak{A}}$ is

$$\lambda u{:}\big[\text{cont:}\textit{ContType}\big]^\frown\big[\text{cont:}\textit{ContType}\big]\ .\ \big[\text{cont:}\mathfrak{S}(\mathfrak{A}(u[0].\text{cont}\mathcal{O}^{\mathfrak{S}}u[1].\text{cont}))\big]$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ T'_{\text{daughter}_1}({}_{\mathcal{O}}T'_{\text{daughter}_2})$ **New!**

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and $\mathcal{O}$ is a combination operation, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ T'_{\text{daughter}_1}({}_{\mathcal{O}}T'_{\text{daughter}_2})$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \underset{\cdot}{\wedge}\ \text{ContForwardApp}_{\mathfrak{S},\mathcal{O}}$$

Note: This covers all the variant notations introduced in previous chapters.

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ B(T'_{\text{daughter}_1}({}_{\mathcal{O}}T'_{\text{daughter}_2}))$ **New!**

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and $\mathcal{O}$ is a combination operation, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ B(T'_{\text{daughter}_1}({}_{\mathcal{O}}T'_{\text{daughter}_2}))$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \underset{\cdot}{\wedge}\ \text{ContForwardApp}_{\mathfrak{S},\mathcal{O},B}$$

$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ \mathfrak{A}(T'_{\text{daughter}_1}({}_{\mathcal{O}}T'_{\text{daughter}_2}))$ **New!**

If T_{mother}, T_{daughter_1} and T_{daughter_2} are sign types and $\mathcal{O}$ is a combination operation, then

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \mid\ \mathfrak{A}(T'_{\text{daughter}_1}({}_{\mathcal{O}}T'_{\text{daughter}_2}))$$

is

$$T_{\text{mother}} \longrightarrow T_{\text{daughter}_1}\ T_{\text{daughter}_2}\ \underset{\cdot}{\wedge}\ \text{ContForwardApp}_{\mathfrak{S},\mathcal{O},\mathfrak{A}}$$

8.4.1.7 Action rules

(as in Chapter 6)

8.4.2 Universal speech act resources

(as in Chapter 2)

8.4.3 Universal discourse resources

$\varphi_1 \mid_{\pi_{11},\pi_{21};\ldots;\pi_{1n},\pi_{2n}} \varphi_2$ **New!**

Suppose that φ_1 and φ_2 and parametric contents, $\pi_{11} \ldots \pi_{1n} \in \mathrm{paths}(\varphi_1.\mathrm{bg})$ and $\pi_{21} \ldots \pi_{2n} \in \mathrm{paths}(\mathcal{F}_{\text{quasi}^*}(\varphi_2.\mathrm{fg})$, then the *content* φ_1 *given* φ_2 *with alignment of* π_{11} *and* π_{21},...,π_{1n} *and* π_{2n}, $\varphi_1 \mid_{\pi_{11},\pi_{21};\ldots;\pi_{1n},\pi_{2n}} \varphi_2$, is

$$\begin{bmatrix} \mathrm{bg} & = & (\varphi_1.\mathrm{bg} \mathbin{\underset{\cdot}{\wedge}} \left[\mathfrak{p}{:}\mathcal{F}_{\text{quasi}^*}(\varphi_2.\mathrm{fg})\right])_{\pi_{11}=\mathfrak{p}.\pi_{21},\ldots,\pi_{1n}=\mathfrak{p}.\pi_{2n}} \\ \mathrm{fg} & = & \lambda c{:}\mathrm{bg}\ .\ \varphi_1(c) \end{bmatrix}$$

$\mathfrak{C}(T_1, T_2)$ **New!**

If T_1 and T_2 are types of parametric contents, then there is a combined type, $\mathfrak{C}(T_1, T_2)$, whose witnesses include witnesses for T_1 given a witness for T_2 with some possible alignment between the two.

The witnesses of $\mathfrak{C}(T_1, T_2)$ are characterized recursively by:

1. if $\varphi : T_1$, then $\varphi : \mathfrak{C}(T_1, T_2)$
2. if $\varphi_1 : \mathfrak{C}(T_1, T_2)$,
   $\pi_{11}, \ldots, \pi_{1n} \in \mathrm{paths}(\varphi_1.\mathrm{bg})$,
   $\varphi_2 : T_2$ and
   $\pi_{21}, \ldots, \pi_{2n} \in \mathrm{paths}(\mathcal{F}_{\text{quasi}^*}(\varphi_2.\mathrm{fg}))$,
   then
   $$\varphi_1|_{\pi_{11},\pi_{21};\ldots;\pi_{1n},\pi_{2n}}\varphi_2 : \mathfrak{C}(T_1, T_2)$$

Interpretation in the context of a previous utterance New! — action rule in example (44).

8.4.4 Universal dialogue resources

(as in Chapter 4)

8.4.5 English resources

8.4.5.1 Types and predicates

(as in Chapter 6)

8.4.5.2 Grammar

Lexical sign types

Let *Lexicon* be the set of lexical sign types defined inductively as follows. The following set is included in *Lexicon*.

{$\mathrm{Lex}_{\mathrm{PropName}}$("Dudamel"),
$\mathrm{Lex}_{\mathrm{PropName}}$("Beethoven"),
$\mathrm{Lex}_{\mathrm{Pron}}$("he"),
$\mathrm{Lex}_{\mathrm{numeral}}$("nine", 9),
$\mathrm{Lex}_{\mathrm{numeral}}$("ninety", 90),
$\mathrm{Lex}_{\mathrm{IndefArt}}$("a"),
$\mathrm{Lex}_{\mathrm{Universal}}$("every"),
$\mathrm{Lex}_{\mathrm{DefArt}}$("the"),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$("composer", *Rec*, composer),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$("conductor", *Rec*, conductor),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$("dog", *Rec*, dog) (= $\Sigma_{\text{“dog”}}$),
RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“dog”}}$), *DogFrame*),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$("passenger", *Rec*, passenger) (= $\Sigma_{\text{“passenger”}}$),
RestrictCommonNoun(CommonNounIndToFrame($\Sigma_{\text{“passenger”}}$), *PassengerFrame*),
$\mathrm{Lex}_{\mathrm{CommonNoun}}$("temperature", *Rec*, temperature) (= $\Sigma_{\text{“temperature”}}$),
RestrictCommonNoun($\Sigma_{\text{“temperature”}}$, *AmbTempFrame*),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$("leave", *Rec*, leave),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$("run", *Rec*, run),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$("rise", $\left[\mathfrak{c}{:}\textit{TempRiseEventCntxt}\right]$, rise),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$("rise", $\left[\mathfrak{c}{:}\textit{PriceRiseEventCntxt}\right]$, rise),
$\mathrm{Lex}_{\mathrm{IntransVerb}}$("rise", $\left[\mathfrak{c}{:}\textit{LocRiseEventCntxt}\right]$, rise),
$\mathrm{Lex}_{\mathrm{TransVerb}}$("hug", *Rec*, hug),
$\mathrm{Lex}_{\mathrm{TransVerb}}$("find", *Rec*, find),
$\mathrm{Lex}_{\mathrm{TransVerb}}$("seek", *Rec*, seek),
$\mathrm{Lex}_{\mathrm{TransVerb}}$("worship", *Rec*, worship),
$\mathrm{Lex}_{\mathrm{be_{ID}}}$("is"),
$\mathrm{Lex}_{\mathrm{be_{scalar}}}$("is"),
Lex("ok", *S*),
Lex("aha", *S*) }

Transitive verbs as verb phrases

If Σ = $\mathrm{Lex}_{\mathrm{TransVerb}}(T_{\mathrm{Phon}}, T_{\mathrm{bg}}, p)$, for some T_{Phon}, T_{bg} and p, and $\Sigma \in$ *Lexicon*, then TransVerbToVerbPhrase(Σ) $\in$ *Lexicon*.

Constituent structure rule components

CnstrIsA **Revised!**

$$\lambda u{:}V\mathbin{\dot\wedge}\big[\text{s-event:}\big[\text{e:“is”}\big]\big]^\frown NP\mathbin{\dot\wedge}\left[\text{syn:}\left[\begin{array}{l}\text{daughters:}Det\mathbin{\dot\wedge}\big[\text{s-event:}\big[\text{e:“a”}\big]\big]\\ \quad {}^\frown N\end{array}\right]\right].$$
$$VP\mathbin{\dot\wedge}\big[\text{cont=}u\text{[2].syn.daughters[2].cont:}\mathit{ContType}\big]$$

Constituent structure rules

Let *CSRules* be the set of constituent structure rules, defined inductively as follows. The following set is included in *CSRules*.

$\{S \longrightarrow NP\ VP \mid \mathcal{B}(NP'(_{@}VP'))$ **Revised!**,
$NP \longrightarrow Det\ N \mid Det'(_{@@}N')$ **Revised!**,
$VP \longrightarrow V\ NP \mathbin{\dot\wedge}$ CnstrIsA,
$VP \longrightarrow V\ NP \mid \mathfrak{A}(V'(_{@}NP'))$ **Revised!**$\}$

Relative clauses

If i and j are natural numbers, then

$Rel \longrightarrow NP_{\mathrm{wh}_i}\ S/j \mid NP'_{\mathrm{wh}_i}(_{@_{\mathrm{wh}_{i,j}}}S/j')$ **Revised!**

is a member of *CSRules*

If T is a type, then

$N \longrightarrow {}^{T}N\,{}^{T}Rel \ \mid\ {}^{T}N'(_{@_{\&}}{}^{T}Rel')$ **Revised!**

is a member of *CSRules*

8.5 Summary

In this chapter we have explored how to use types to characterize underspecified interpretation. A type underspecifies its witnesses since any one of them can be used as a specific object of the type. We thus characterize types of contents to play the role of underspecified representations as employed, for example, in DRT by Reyle (1993) or using Quasi Logical Form (Alshawi, 1992). Using underspecified representations of meaning is important in computational applications where a large number of alternative specified representations can be generated. The idea here is that we can compute a single type of content for an utterance and then determine a precise content as needed and check that it is of the type generated. This helps us understand how speakers of a natural language can rapidly process utterances in real time when in fact many different contents are available.

We started by recreating the storage algorithm of Cooper (1983), showing how the stores for quantifiers can be incorporated into context on the structure view of context we have developed using record types. We then showed how a treatment of anaphora can be added to the elaboration of content types including discourse anaphora, donkey anaphora and aspects of Chomskian binding theory.

While we do not claim that we have covered everything that has been accounted for in the considerable linguistic literature on these topics, there does seem to be enough here to show that our theory of types which we used in the earlier parts of the book to deal with perception, interaction and mental states, is also capable of dealing with these central concerns of linguistic semantics.

Conclusion

What view of language have we come to after working through the details of this book?

Appendix

TTR

Unless otherwise stated this is the version of TTR presented in Cooper (2012b).

1 Underlying set theory

In previous statements of this system such as Cooper (2012b) we tacitly assumed a standard underlying set theory such as ZF (Zermelo-Fraenkel) with urelements (as formulated for example in Suppes, 1960). This is what we take to be the common or garden working set theory which is familiar from the core literature on formal semantics deriving from Montague's original work (Montague, 1974).

In this version we will assume that our set theory comes equipped with a set of *urelements* (entities which are not sets but which can be members of sets). We will assume that among the urelements is a countably infinite set which is designated as the set of *labels*. In addition we assume that among the urelements there is a finite or countably infinite set, disjoint from the set of labels, designated as the set of *flavours*. An *(unflavoured) labelled set* (Chapter 1, p. 21) is a set of ordered pairs whose first member is a label and whose second element is either an urelement which is not a label or a set (possibly a labelled set), such that no more than one ordered pair can contain any particular label as its first member. This means that a labelled set is the traditional set theoretic construction of an extensional function from a set of labels onto some set. A *flavoured labelled set* is an unflavoured labelled set, X, with the addition of some flavour, $\mathfrak{f}$, that is, $X \cup \{\mathfrak{f}\}$. We use flavours when we need to distinguish objects which correspond to the same set of ordered pairs.

We refer to the first members of the pairs in a labelled set (flavoured or unflavoured) as *labels* used in the labelled set and we will refer to the second members of the ordered pairs as the *labelled elements* of the labelled set. If X is a labelled set we use $\mathrm{labels}(X)$ to represent the set $\{\ell \mid \exists x\ \langle \ell, x\rangle \in X\}$, the left projection of X. If $\ell \in \mathrm{labels}(X)$ and $\langle \ell, v\rangle \in X$ then we use $X.\ell$ to represent v. We characterize the set of *paths* in a labelled set, $\mathrm{paths}(X)$ by the following inductive definition:

If X is a labelled set, then

1. if $\ell \in \mathrm{labels}(X)$, then $\ell \in \mathrm{paths}(X)$
2. if $\ell \in \mathrm{labels}(X)$, $X.\ell$ is a labelled set and $\pi \in \mathrm{paths}(X.\ell)$, then $\ell.\pi \in \mathrm{paths}(X)$

The set of *total paths* in a labelled set, X, $\mathrm{tpaths}(X)$ is the set of paths, π, such that $\pi \in \mathrm{paths}(X)$ and $X.\pi$ is not a labelled set.

π_1 is an *initial subpath* (Chapter 4, p. 178) of π_2, $\pi_1 \leq \pi_2$, just in case either $\pi_1 = \pi_2$ or there is some π such that $\pi_2 = \pi_1.\pi$.

π_1 is a *proper initial subpath* (Chapter 4, p. 178) of π_2, $\pi_1 < \pi_2$, just in case there is some π such that $\pi_2 = \pi_1.\pi$.

We characterize *the subtraction of a path, π, from a labelled set X*, $X \ominus \pi$, (Chapter 7, p. 388) as follows:

1. If X is a labelled set, $\ell \in \mathrm{labels}(X)$ and $\langle \ell, \varphi \rangle \in X$, then $X \ominus \ell$ is

   $X - \{\langle \ell, \varphi \rangle\}$

2. If X is a labelled set, $\langle \ell, \varphi \rangle \in X$ and $\ell.\pi \in \mathrm{tpaths}(X)$, then

   if $\ell.\pi$ is branching in X, then $X \ominus \ell.\pi$ is

   $(X - \{\langle \ell, \varphi \rangle\}) \cup \{\langle \ell, \varphi \ominus \pi \rangle\}$

   otherwise $X \ominus \ell.\pi$ is

   $X - \{\langle \ell, \varphi \rangle\}$

If X is a labelled set and $\pi_1, \ldots, \pi_n \in \mathrm{tpaths}(X)$, then we write $X \ominus \pi_1, \ldots, \pi_n$ for $X \ominus \pi_1 \ominus \ldots \ominus \pi_n$.

2 Basic types

System of basic types

A *system of basic types* (Chapter 1, p. 14) is a pair:

$\textbf{TYPE}_B = \langle \textbf{Type}, A \rangle$

where:

1. **Type** is a non-empty set
2. A is a function whose domain is **Type**
3. for any $T \in$ **Type**, $A(T)$ is a set disjoint from **Type**
4. for any $T \in$ **Type**, $a :_{\mathbf{TYPE_B}} T$ iff $a \in A(T)$

3 Complex types

3.1 Predicates

We start by introducing the notion of a predicate signature.

A *predicate signature* (Chapter 1, p. 16) is a triple

$\langle$**Pred**, **ArgIndices**, *Arity*$\rangle$

where:

1. **Pred** is a set (of predicates)
2. **ArgIndices** is a set (of indices for predicate arguments, normally types)
3. *Arity* is a function with domain **Pred** and range included in the set of finite sequences of members of **ArgIndices**.

A *polymorphic predicate signature* (Chapter 1, p. 17) is a triple

$\langle$**Pred**, **ArgIndices**, *Arity*$\rangle$

where:

1. **Pred** is a set (of predicates)

2. **ArgIndices** is a set (of indices for predicate arguments, normally types)
3. *Arity* is a function with domain **Pred** and range included in the powerset of the set of finite sequences of members of **ArgIndices**.

3.2 Systems of complex types

A *system of complex types* (Chapter 1, p. 18) is a quadruple:

TYPE$_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle$⟩

where:

1. ⟨**BType**, A⟩ is a system of basic types
2. **BType**⊆**Type**
3. for any $T \in$ **Type**, if $a :_{\langle \mathbf{BType}, A\rangle} T$ then $a :_{\mathbf{TYPE_C}} T$
4. ⟨**Pred**, **ArgIndices**, *Arity*⟩ is a (polymorphic) predicate signature
5.[1] $P(a_1, \ldots a_n) \in$ **PType** iff $P \in$ **Pred**, $T_1 \in \mathbf{Type}, \ldots, T_n \in \mathbf{Type}$, *Arity*($P$)=$\langle T_1, \ldots, T_n\rangle$ ($\langle T_1, \ldots, T_n\rangle \in$*Arity*($P$)) and $a_1 :_{\mathbf{TYPE_C}} T_1, \ldots, a_n :_{\mathbf{TYPE_C}} T_n$
6. **PType**⊆**Type**
7. for any $T \in$ **PType**, $F(T)$ is a set disjoint from **Type**
8. for any $T \in$ **PType**, $a :_{\mathbf{TYPE_C}} T$ iff $a \in F(T)$

We call the pair $\langle A, F\rangle$ in a complex system of types the *model* because of its similarity to first order models in providing values for the basic types and the ptypes constructed from predicates and arguments. It is this pair which connects the system of types to the non-type theoretical world of objects and situations.

In Cooper (2012b) we did not define exactly what entity is represented by $P(a_1, \ldots a_n)$. Here we will specify it to be the labelled set

$\{\langle \mathrm{pred}, P\rangle, \langle \mathrm{arg}_1, a_1\rangle, \ldots, \langle \mathrm{arg}_n, a_n\rangle\}$

where ‘pred’, ‘arg$_i$’ are reserved labels (not used except as required here).

[1] This clause has been modified since Cooper (2012b) where it was a conditional rather than a biconditional.

4 Function types

A system of complex types $\mathbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has function types* (Chapter 1, p. 29) if

1. for any $T_1, T_2 \in$ **Type**, $(T_1 \to T_2) \in$ **Type**
2. for any $T_1, T_2 \in$ **Type**, $f :_{\mathbf{TYPE_C}} (T_1 \to T_2)$ iff f is a function whose domain is $\{a \mid a :_{\mathbf{TYPE_C}} T_1\}$ and whose range is included in $\{a \mid a :_{\mathbf{TYPE_C}} T_2\}$

In Cooper (2012b) we did not specify exactly what object is represented by a function type $(T_1 \to T_2)$. Here we specify it to be the labelled set

$\{\langle \text{dmn}, T_1\rangle, \langle \text{rng}, T_2\rangle\}$

where 'dmn' ("domain") and 'rng' ("range") are reserved labels.

We also introduce a limited kind of polymorphism is function types which we did not have in Cooper (2012b).

A system of complex types $\mathbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ with function types *has partial function types* (Chapter 2, p. 63) if

1. for any $T_1, T_2 \in$ **Type**, $(T_1 \rightharpoonup T_2) \in$ **Type**
2. for any $T_1, T_2 \in$ **Type**, $f :_{\mathbf{TYPE_C}} (T_1 \rightharpoonup T_2)$ iff there is some type T' such that $f : (T' \to T_2)$ and for any a, if $a : T'$ then $a : T_1$

We specify the type $(T_1 \rightharpoonup T_2)$ to be the labelled set

$\{\langle \text{partdmn}, T_1\rangle, \langle \text{rng}, T_2\rangle\}$

where 'partdmn' ("partial domain") and 'rng' ("range") are reserved labels ('rng' being the same reserved label that was used for total function types).

We introduce a notation for functions which is borrowed from the λ-calculus:

$\lambda v{:}T \,.\, \varphi$ (Chapter 1, p. 30) is that function f such that for any $a : T$, $f(a)$ (the result of applying f to a) is represented by $\varphi[v \leftarrow a]$ (the result of replacing any free occurrence of v in φ with a).

For example, the graph of the function

λv:*Ind* . run(v)

is the set of ordered pairs

$\{\langle v, \mathrm{run}(v)\rangle \mid v : \mathit{Ind}\}$

Recall that 'run(v)' is itself a representation for the labelled set

$\{\langle \mathrm{pred}, \mathrm{run}\rangle, \langle \mathrm{arg}_1, v\rangle\}$

Note that if f is the function λv:*Ind* . run(v) and a:*Ind* then $f(a)$ (the result of applying f to a) is 'run(a)'. Our definition of function-argument application guarantees what is called β-equivalence in the λ-calculus. When we discuss record types as arguments to functions we will need to introduce one slight complication to our notion of function application. We will introduce that complication when we discuss record types.

In order to maintain α-equivalence (for example, if v_1 and v_2 are distinct variables, $\lambda v_1 : T \, . \, \varphi(\!(v_1)\!)$ represents the same function as $\lambda v_2 : T \, . \, \varphi(\!(v_2)\!)$), we use a variant of de Bruijn indexing (de Bruijn, 1972) in the labelled set we use to model functions. We model functions as labelled sets with two fields with the distinguished labels 'lambda' and 'body'. Where de Bruijn uses natural numbers to index the variable positions we use segments of paths in the labelled set which relate the appropriate instance of the 'body'-label labelling the field which is sister to the binding 'lambda'-field. Thus

> λv:*Ind* . run(v)

is the labelled set

> {⟨lambda, *Ind*⟩,
> ⟨body, {⟨pred, run⟩,
> ⟨arg_1, body.arg_1⟩}⟩}

and

> λv_1:*Ind* . $r_1(v_1, \lambda v_2$:*Ind* . $r_2(v_2, v_1))$

is the labelled set

> {⟨lambda, *Ind*⟩,

$\langle$body, $\{\langle$pred, $r_1\rangle$,
$\langle$arg$_1$, body.arg$\rangle$,
$\langle$arg$_2$, $\{\langle$lambda, *Ind*$\rangle$,
$\langle$body, $\{\langle$pred, $r_2\rangle$,
$\langle$arg$_1$, body.arg$_1\rangle$,
$\langle$arg$_2$, body.arg$_2$.body.arg$_2\rangle\}\rangle\}\rangle\}\rangle\}$

5 Set types

Set types were not included in Cooper (2012b).

A system of complex types **TYPE**$_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ *has set types* (Chapter 2, p. 65) if

1. for any $T \in$ **Type**, $\mathrm{set}(T) \in$ **Type**
2. for any $T \in$ **Type**, $X :_{\mathbf{TYPE_C}} \mathrm{set}(T)$ iff X is a set and for all $a \in X$, $a :_{\mathbf{TYPE_C}} T$

We let $\mathrm{set}(T)$ represent the labelled set $\{\langle \mathrm{set}, T\rangle\}$ where 'set' is a reserved label.

@@

If T is a type and $X : \mathrm{set}(T)$, then $\mathfrak{T}(X)$ is a type. $a : \mathfrak{T}(X)$ iff $a \in X$. (Chapter 7, p. 356)

Need to expand this to a system of complex types with set types and membership types

In order to define plurality types as subtypes of set types we introduce a notion of one record being a proper part of another.

For any objects r_1 and r_2, r_1 *is a proper part of* r_2, $r_1 < r_2$, (Chapter 5, p. 262) just in case

1. r_1 and r_2 are records,
2. $\mathrm{paths}_{\mathrm{rec}}(r_1) \subset \mathrm{paths}_{\mathrm{rec}}(r_2)$ and
3. for all $\pi \in \mathrm{tpaths}(r_1)$, $r_1.\pi = r_2.\pi$

A system of complex types **TYPE**$_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ with set types *has plurality types* (Chapter 5, p. 262) if

1. for any $T \in$ **Type**, $\mathrm{plurality}(T) \in$ **Type**

2. for any $T \in$ **Type**, $A :_{\mathbf{TYPE_C}} \text{plurality}(T)$ iff
   a) $A :_{\mathbf{TYPE_C}} \text{set}(T)$
   b) if $a \in A$ then for any b such that $a < b$, $b \notin A$

We let $\text{plurality}(T)$ represent the labelled set $\{\langle\text{plurality}, T\rangle\}$ where 'plurality' is a reserved label.

6 Singleton types

Singleton types were not included in the formal definition in Cooper (2012b).

A system of complex types $\mathbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle$⟩ *has singleton types* (Chapter 2, p. 59) if

1. for any $T, T' \in$ **Type** and $a :_{\mathbf{TYPE_C}} T'$, $T_a \in$ **Type**
2. for any $T, T' \in$ **Type** and $a :_{\mathbf{TYPE_C}} T'$, $b :_{\mathbf{TYPE_C}} T_a$ iff $b :_{\mathbf{TYPE_C}} T$ and $a = b$

We let T_a represent the labelled set $\{\langle\text{singleton}, \langle T, a\rangle\rangle\}$ where 'singleton' is a reserved label.

7 Join types

A system of complex types $\mathbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle$⟩ *has join types* (Chapter 2, p. 64) if

1. for any $T_1, T_2 \in \mathbf{Type}$, $(T_1 \vee T_2) \in \mathbf{Type}$
2. for any $T_1, T_2 \in \mathbf{Type}$, $a :_{\mathbf{TYPE_C}} (T_1 \vee T_2)$ iff $a :_{\mathbf{TYPE_C}} T_1$ or $a :_{\mathbf{TYPE_C}} T_2$

Here, but not in Cooper (2012b), we specify that $(T_1 \vee T_2)$ represents the labelled set $\{\langle\text{disj}_1, T_1\rangle, \langle\text{disj}_2, T_2\rangle\}$ where 'disj$_1$' and 'disj$_2$' are reserved labels ("disjunct").

We add generalized join types which were not present in Cooper (2012b). A system of complex types $\mathbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle$⟩ *has generalized join types* (Chapter 2, p. 65) if

1. for any finite set of types, $\mathcal{T}$, such that $\mathcal{T} \subseteq \mathbf{Type}$, $\bigvee \mathcal{T} \in \mathbf{Type}$

2. for any finite $\mathcal{T} \subseteq \mathbf{Type}$, $a :_{\mathbf{TYPE_C}} \bigvee \mathcal{T}$ iff $a :_{\mathbf{TYPE_C}} T$ for some $T \in \mathcal{T}$

We specify that $\bigvee \mathcal{T}$ represents the labelled set $\{\langle \mathrm{disj}, \mathcal{T}\rangle\}$ where 'disj' is a reserved label ("disjunction").

8 Meet types

A system of complex types **TYPE**$_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has meet types* (Chapter 2, p. 86) if

1. for any $T_1, T_2 \in \mathbf{Type}$, $(T_1 \wedge T_2) \in \mathbf{Type}$
2. for any $T_1, T_2 \in \mathbf{Type}$, $a :_{\mathbf{TYPE_C}} (T_1 \wedge T_2)$ iff $a :_{\mathbf{TYPE_C}} T_1$ and $a :_{\mathbf{TYPE_C}} T_2$

Here, but not in Cooper (2012b), we specify that $(T_1 \wedge T_2)$ represents the labelled set $\{\langle \mathrm{conj}_1, T_1\rangle, \langle \mathrm{conj}_2, T_2\rangle\}$ where 'conj$_1$' and 'conj$_2$' are reserved labels ("conjunct").

We add generalized meet types which were not present in Cooper (2012b). A system of complex types **TYPE**$_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle\rangle$ *has generalized meet types* (Chapter 2, p. 87) if

1. for any non-empty finite set of types, $\mathbb{T}$, such that $\mathbb{T} \subseteq \mathbf{Type}$, $\bigwedge \mathbb{T} \in \mathbf{Type}$
2. for any finite $\mathbb{T} \subseteq \mathbf{Type}$, $a :_{\mathbf{TYPE_C}} \bigwedge \mathbb{T}$ iff $a :_{\mathbf{TYPE_C}} T$ for all $T \in \mathbb{T}$

We specify that $\bigwedge \mathbb{T}$ represents the labelled set $\{\langle \mathrm{conj}, \mathbb{T}\rangle\}$ where 'conj' is a reserved label ("conjunction").

9 Models and modal systems of types

A modal system of complex types provides a collection of models, $\mathcal{M}$, so that we can talk about properties of the whole collection of type assignments provided by the various models $M \in \mathcal{M}$.

A *modal system of complex types based on* $\mathcal{M}$ (Chapter 1, p. 40) is a family of quadruples[2]:

TYPE$_{MC}$ = ⟨**Type**$_M$, **BType**, ⟨**PType**$_M$, **Pred**, **ArgIndices**, *Arity*⟩, $M\rangle_{M \in \mathcal{M}}$

[2]This definition has been modified since Cooper (2012b) to make **PType** and **Type** be relativized to the model M.

where for each $M \in \mathcal{M}$, $\langle$**Type**$_M$, **BType**, $\langle$**PType**$_M$, **Pred**, **ArgIndices**, *Arity*$\rangle$, $M\rangle$ is a system of complex types.

This enables us to define modal notions:

If **TYPE**$_{MC}$ = $\langle$**Type**$_M$, **BType**, $\langle$**PType**$_M$, **Pred**, **ArgIndices**, *Arity*$\rangle$, $M\rangle_{M\in\mathcal{M}}$ is a modal system of complex types based on $\mathcal{M}$, we shall use the notation **TYPE**$_{MC_M}$ (where $M \in \mathcal{M}$) to refer to that system of complex types in **TYPE**$_{MC}$ whose model is M. Let **Type**$_{MC_{restr}}$ be $\bigcap_{M\in\mathcal{M}}$ **Type**$_M$, the "restrictive" set of types which occur in all possibilities, and **Type**$_{MC_{incl}}$ be $\bigcup_{M\in\mathcal{M}}$ **Type**$_M$, the "inclusive" set of types which occur in at least one possibility. Then we can define modal notions either restrictively or inclusively (indicated by the subscripts r and i respectively):

restrictive modal notions

1. for any $T_1, T_2 \in$ **Type**$_{MC_{restr}}$, T_1 *is (necessarily) equivalent$_r$ to* T_2 *in* **TYPE**$_{MC}$, $T_1 \approx_{\mathbf{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$, $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_1\} = \{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_2\}$
2. for any $T_1, T_2 \in$ **Type**$_{MC_{restr}}$, T_1 *is a subtype$_r$ of* T_2 *in* **TYPE**$_{MC}$, $T_1 \sqsubseteq_{\mathbf{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$, $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_1\} \subseteq \{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_2\}$
3. for any $T \in$ **Type**$_{MC_{restr}}$, T *is necessary$_r$ in* **TYPE**$_{MC}$ iff for all $M \in \mathcal{M}$, $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T\} \neq \emptyset$
4. for any $T \in$ **Type**$_{MC_{restr}}$, T *is possible$_r$ in* **TYPE**$_{MC}$ iff for some $M \in \mathcal{M}$, $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T\} \neq \emptyset$

inclusive modal notions

1. for any $T_1, T_2 \in$ **Type**$_{MC_{incl}}$, T_1 *is (necessarily) equivalent$_i$ to* T_2 *in* **TYPE**$_{MC}$, $T_1 \approx_{\mathbf{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$, if T_1 and T_2 are members of **Type**$_M$, then $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_1\} = \{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_2\}$
2. for any $T_1, T_2 \in$ **Type**$_{MC_{incl}}$, T_1 *is a subtype$_i$ of* T_2 *in* **TYPE**$_{MC}$, $T_1 \sqsubseteq_{\mathbf{TYPE}_{MC}} T_2$, iff for all $M \in \mathcal{M}$, if T_1 and T_2 are members of **Type**$_M$, then $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_1\} \subseteq \{a \mid a :_{\mathbf{TYPE}_{MC_M}} T_2\}$
3. for any $T \in$ **Type**$_{MC_{incl}}$, T *is necessary$_i$ in* **TYPE**$_{MC}$ iff for all $M \in \mathcal{M}$, if $T \in$**Type**$_M$, then
   $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T\} \neq \emptyset$
4. for any $T \in$ **Type**$_{MC_{incl}}$, T *is possible$_i$ in* **TYPE**$_{MC}$ iff for some $M \in \mathcal{M}$, if $T \in$**Type**$_M$, then
   $\{a \mid a :_{\mathbf{TYPE}_{MC_M}} T\} \neq \emptyset$

It is easy to see that if any of the restrictive definitions holds for given types in a particular system then the corresponding inclusive definition will also hold for those types in that system.

10 The type *Type* and stratification

An *intensional system of complex types* (Chapter 1, p. 32) is a family of quadruples indexed by the natural numbers:

$$\textbf{TYPE}_{IC} = \langle \textbf{Type}^n, \textbf{BType}, \langle \textbf{PType}^n, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \langle A, F^n\rangle\rangle_{n\in Nat}$$

where (using $\mathbf{TYPE}_{IC_n}$ to refer to the quadruple indexed by n):

1. for each n,$\langle \textbf{Type}^n, \textbf{BType}, \langle \textbf{PType}^n, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \langle A, F^n\rangle\rangle$ is a system of complex types
2. for each n, $\mathbf{Type}^n \subseteq \mathbf{Type}^{n+1}$ and $\mathbf{PType}^n \subseteq \mathbf{PType}^{n+1}$
3. for each n, if $T \in \mathbf{PType}^n$ then $F^n(T) \subseteq F^{n+1}(T)$
4. for each $n > 0$, $\textit{Type}^n \in \mathbf{Type}^n$
5. for each $n > 0$, $T :_{\mathbf{TYPE}_{IC_n}} \textit{Type}^n$ iff $T \in \mathbf{Type}^{n-1}$

Here, but not in Cooper (2012b), we make explicit that *Type* is a distinguished urelement and that *Type*^{n} represents the labelled set $\{\langle \mathrm{ord}, n\rangle, \langle \mathrm{typ}, \textit{Type}\rangle\}$ where 'ord' and 'typ' are reserved labels ("order", "type").

An intensional system of complex types $\textbf{TYPE}_{IC}$,

$$\textbf{TYPE}_{IC} = \langle \textbf{Type}^n, \textbf{BType}, \langle \textbf{PType}^n, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \langle A, F^n\rangle\rangle_{n\in Nat}$$

has dependent function types (Chapter 2, p. 61) if

1. for any $n > 0$, $T \in \mathbf{Type}^n$ and $\mathcal{T} :_{\mathbf{TYPE}_{IC_n}} (T \rightarrow \textit{Type}^n)$, $((a : T) \rightarrow \mathcal{T}(a)) \in \mathbf{Type}^n$
2. for each $n > 0$, $f :_{\mathbf{TYPE}_{IC_n}} ((a : T) \rightarrow \mathcal{T}(a))$ iff f is a function whose domain is $\{a \mid a :_{\mathbf{TYPE}_{IC_n}} T\}$ and such that for any a in the domain of f, $f(a) :_{\mathbf{TYPE}_{IC_n}} \mathcal{T}(a)$.

We might say that on this view dependent function types are "semi-intensional" in that they depend on there being a type of types for their definition but they do not introduce types as arguments to predicates and do not involve the definition of orders of types in terms of the types of the next lower order.

Here, in contrast to Cooper (2012b), we make explicit that $((a : T) \to \mathcal{F}(a))$ represents the labelled set $\{\langle \mathrm{dmn}, T\rangle, \langle \mathrm{deprng}, \mathcal{F}\rangle\}$ where 'dmn' as before for function types is a reserved label corresponding to "domain" and 'deprng' is a reserved label corresponding to "dependent range".

Let M be a model, $\langle A, F\rangle$, where A is an assignment to basic types and F an assignment to ptypes as usual. Let $\mathcal{M}$ be an infinite sequence of models, M, indexed by the natural numbers, corresponding to the models for the type systems of each order in an intensional type system. We use $\mathcal{M}_n$ to represent the model for the n-th order in an intensional type system. We use $\mathfrak{M}$ to represent a set of such model sequences, representing the model sequences for each of the possibilities in the intensional modal type system. Putting the definition of a modal type system and an intensional type system together we obtain:[3]

An *intensional modal system of complex types based on* $\mathfrak{M}$ (Chapter 6, p. 291) is a family, indexed by the natural numbers, of families of quadruples indexed by members of $\mathfrak{M}$:

$$\textbf{TYPE}_{IMC} = \langle \textbf{Type}^n, \textbf{BType}, \langle \textbf{PType}^n, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \mathcal{M}_n\rangle_{\mathcal{M}\in\mathfrak{M}, n\in Nat}$$

where:

1. for each n, $\langle \textbf{Type}^n, \textbf{BType}, \langle \textbf{PType}^n, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \mathcal{M}_n\rangle_{\mathcal{M}\in\mathfrak{M}}$ is a modal system of complex types based on $\{\mathcal{M}_n \mid \mathcal{M} \in \mathfrak{M}\}$
2. for each $\mathcal{M} \in \mathfrak{M}$, $\langle \textbf{Type}^n, \textbf{BType}, \langle \textbf{PType}^n, \textbf{Pred}, \textbf{ArgIndices}, \textit{Arity}\rangle, \mathcal{M}_n\rangle_{n\in Nat}$ is an intensional system of complex types

11 Records and Record types

In this section we will define what it means for a system of complex types to have record types. The objects of record types, that is, records, are themselves structured mathematical objects of a particular kind and we will start by characterizing them.

[3]This explicit definition was not present in Cooper (2012b).

11.1 Records

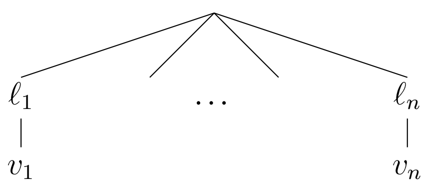
r is a *record according to a set of labels $\mathcal{L}$ and a type system* $\mathbb{T}$ (Chapter 1, p. 33) iff r is a finite labelled set (Appendix 1) whose labels are included in $\mathcal{L}$ and for any labelled element, v, in r, there is some type T such that $v :_{\mathbb{T}} T$.

If r is a record and $\langle \ell, v \rangle$ is in r, we call $\langle \ell, v \rangle$ a *field* of r, ℓ a *label* in r and v a *value* in r (the *value of ℓ in r*). We use $r.\ell$ to denote v.

We use a tabular format to represent records. A record $\{\langle \ell_1, v_1 \rangle, \ldots, \langle \ell_n, v_n \rangle\}$ is displayed as

$$\left[\begin{array}{ccc} \ell_1 & = & v_1 \\ & \vdots & \\ \ell_n & = & v_n \end{array}\right]$$

An alternative notation is as a tree (or directed graph) whose root is unlabelled:



We will sometime says that the occurrences $\ell_1, \ldots, \ell_n$ are *sisters* in the record.

Records, as labelled sets, will have paths as defined for labelled sets. However, we will want in addition a more restricted notion of path for records which excludes those paths which include the distinguished labels used in non-record structures which may be values in a record (Chapter 1,p. 33).

If r is a record, then

1. if $\ell \in \text{labels}(r)$, then $\ell \in \text{paths}_{\text{rec}}(r)$
2. if $\ell \in \text{labels}(r)$, $r.\ell$ is a record and $\pi \in \text{paths}_{\text{rec}}(r.\ell)$, then $\ell.\pi \in \text{paths}_{\text{rec}}(r)$

Similarly, $\text{tpaths}_{\text{rec}}(r)$ is the set of paths, π, such that $\pi \in \text{paths}_{\text{rec}}(r)$ and $r.\pi$ is not a record.

We sometimes use 'paths' and 'tpaths' without the subscript for these more restricted notions when there is no risk for confusion.

A value may itself be a record and paths may extend into embedded records. A record which contains records as values is called a *complex record* and otherwise a record is *simple*. Values which are not records are called *leaves*. Consider a record r

$$\begin{bmatrix} f & = & \begin{bmatrix} f & = & \begin{bmatrix} \mathit{ff} & = & a \\ gg & = & b \end{bmatrix} \\ g & = & c \end{bmatrix} \\ g & = & \begin{bmatrix} h & = & \begin{bmatrix} g & = & a \\ h & = & d \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Among the paths in r are $r.f$, $r.g.h$ and $r.f.f.\mathit{ff}$ which denote, respectively,

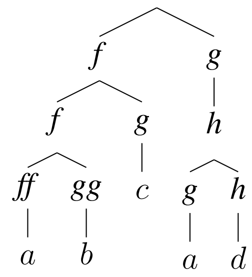
$$\begin{bmatrix} f & = & \begin{bmatrix} \mathit{ff} & = & a \\ gg & = & b \end{bmatrix} \\ g & = & c \end{bmatrix}$$

$$\begin{bmatrix} g & = & a \\ h & = & d \end{bmatrix}$$

and a. We will make a distinction between *absolute paths*, such as those we have already mentioned, which consist of a record followed by a series of labels connected by dots and *relative paths* which are just a series of labels connected by dots, e.g. $g.h$. Relative paths are useful when we wish to refer to similar paths in different records. We will use *path* to refer to either absolute or relative paths when it is clear from the context which is meant. The set of leaves of r, also known as its *extension* (those objects other than labels which it contains), is $\{a, b, c, d\}$. The bag (or multiset) of leaves of r, also known as its *multiset extension*, is $\{a, a, b, c, d\}$. A record may be regarded as a way of labelling and structuring its extension.

An object, a, is a *component* (Chapter 3, p. 127) of a record, r, in symbols, $a\varepsilon r$, just in case there is some path, π, in r such that $r.\pi = a$. Thus the record, r, above has the following components: $r.f$, $r.f.f$, $r.f.f.\mathit{ff}$, $r.f.f.gg$, $r.f.g$, $r.g$, $r.g.h$, $r.g.h.g$ and $r.g.h.h$. An object, a, is *present* (Chapter 3, p. 128) in a record, r, in symbols, $a\underline{\varepsilon}r$, just in case either $a = r$ or $a\varepsilon r$.

We can, of course, also think of complex records as trees, for example:



We can thus use tree terminology to describe relationships between occurrences of labels. For example, we can say that the topmost occurrence of f immediately dominates another occurrence of f and an occurrence of g.

Two records are *(multiset) extensionally equivalent* if they have the same (multiset) extension. Two important, though trivial, facts about records are:

Flattening. For any record r, there is a multiset extensionally equivalent simple record. We can define an operation of flattening on records which will always produce an equivalent simple record. In the case of our example, the result of flattening is

$$\begin{bmatrix} f.f.ff & = & a \\ f.f.gg & = & b \\ f.g & = & c \\ g.h.g & = & a \\ g.h.h & = & d \end{bmatrix}$$

assuming the flattening operation uses paths from the original record in a rather obvious way to create unique labels for the new record.

Relabelling. For any record r, if $\pi_1.\ell.\pi_2$ is a path π in r, and $\pi_1.\ell'.\pi_2{}'$ is *not* a path in r (for any $\pi_2{}'$), then substituting ℓ' for the occurrence of ℓ in π results in a record which is multiset equivalent to r. We could, for example, substitute k for the second occurrence of g in the path $g.h.g$ in our example record.

$$\begin{bmatrix} f & = & \begin{bmatrix} f & = & \begin{bmatrix} ff & = & a \\ gg & = & b \end{bmatrix} \\ g & = & c \end{bmatrix} \\ g & = & \begin{bmatrix} h & = & \begin{bmatrix} k & = & a \\ h & = & d \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

We will first look in more detail at flattening. Flattened records have *complex labels* (or *paths*) of the form $\ell_1.\ell_2.\ldots.\ell_n$. If L is a set of labels, the set of *complex labels*, L_π, *based on* L is the minimal set such that:

1. if $\ell \in L$, then $\ell \in L_\pi$
2. if $\pi \in L_\pi$ and $\ell \in L$, then $\pi.\ell \in L_\pi$

Recall that a record, r, is a set of fields, f, where each f, is an ordered pair, $\langle \ell, v\rangle$ where ℓ is a (complex) label and v is the value of the field which may itself be a record or some other object. If r is a record, then $\varphi(r)$, *the flattening of* r, is

$$\bigcup_{f \in r} \text{FlattenField}(f)$$

where $\text{FlattenField}(\langle \ell, v\rangle)$ is

$$\varphi(\{\langle \ell.\ell_1, v_1\rangle, \langle \ell.\ell_2, v_2\rangle, \ldots, \langle \ell.\ell_n, v_n\rangle\})$$

if v is the record $\{\langle \ell_1, v_1\rangle, \langle \ell_2, v_2\rangle, \ldots, \langle \ell_n, v_n\rangle\}$ and $\langle \ell, v\rangle$ otherwise.

We can show that if r is a record then $\varphi(r)$ is a record. It follows trivially from the requirement that records are graphs of functions from labels to values. If r is a record $\{\langle \ell_1, v_1\rangle, \ldots, \langle \ell_n, v_n\rangle\}$, then the *set of labels of* r, $\text{labels}(r)$, is $\{\ell_1, \ldots, \ell_n\}$. For example, if r is the record

$$\left[\begin{array}{lll} \text{x} & = & \left[\begin{array}{lll} \text{z} & = & a \\ \text{w} & = & b \end{array}\right] \\ \text{y} & = & c \end{array}\right]$$

then $\text{labels}(r)$ is $\{\text{x}, \text{y}\}$. That is, 'labels' picks out the "highest level" of labels in a complex record. We use $\text{tpaths}(r)$ to represent the *set of total paths* of r, $\text{labels}(\varphi(r))$.

If L is a set of labels, then the set of *complex labels* (or *paths*), L_+, based on L is the minimal set such that

1. if $\ell \in L$, then $\ell \in L_+$
2. if $\pi \in L_+$ and $\ell \in L$, then $\pi.\ell \in L_+$

If r is a record such that $\text{labels}(r) \subseteq L_+$, then the *inverse flattening* (or *expansion*) *of* r, $\varphi^-(r)$, is the minimal set such that

1. if $\langle \ell, v \rangle \in r$ and $\ell \in L$, then $\langle \ell, v \rangle \in \varphi^{-}(r)$
2. if $\ell \in L$, $\pi \in L_{+}$ and $\langle \ell.\pi, v \rangle \in r$, then $\langle \ell, \varphi^{-}(\{\langle \pi', v' \rangle \mid \langle \ell.\pi', v' \rangle \in r\}) \rangle \in \varphi^{-}(r)$

Claim: If r is a record, then $\varphi^{-}(\varphi(r)) = r$.

The *set of total paths of* r, $\mathrm{tpaths}(r)$, is $\mathrm{labels}(\varphi(r))$.

If r is a record, then the *set of paths of* r, $\mathrm{paths}(r)$, is the minimal set such that

1. If $\pi \in \mathrm{tpaths}(r)$, then $\pi \in \mathrm{paths}(r)$
2. If ℓ is a label (in L), π is a non-empty path based on L and $\pi.\ell \in \mathrm{paths}(r)$, then $\pi \in \mathrm{paths}(r)$

If Π is a set of paths (in L_{+}), then the *set of longest paths in* Π, $\mathrm{longest}(\Pi)$ is such that $\pi \in \mathrm{longest}(\Pi)$ iff $\pi \in \Pi$ and there is no π' such that $\pi.\pi' \in \Pi$.

Claim: $\mathrm{longest}(\mathrm{paths}(r)) = \mathrm{tpaths}(r)$

If R is a set of records, then the *set of common paths of* R, $\mathrm{cpaths}(R)$, is

$$\bigcap_{r \in R} \mathrm{paths}(r),$$

the *set of longest common paths of* R, $\mathrm{lcpaths}(R)$, is

$$\mathrm{longest}(\bigcap_{r \in R} \mathrm{paths}(r))$$

and the *set of common total paths of* R, $\mathrm{ctpaths}(R)$, is

$$\bigcap_{r \in R} \mathrm{tpaths}(r)$$

Claim: $\mathrm{lcpaths}(R) = \mathrm{ctpaths}(R)$

Consider the records

$$r_1 = \left[\begin{array}{lll} \mathrm{x} & = & \left[\begin{array}{lll} \mathrm{x} & = & a \\ \mathrm{y} & = & b \end{array}\right] \\ \mathrm{y} & = & \left[\begin{array}{lll} \mathrm{x} & = & c \\ \mathrm{y} & = & d \end{array}\right] \end{array}\right]$$

$$r_2 = \left[\begin{array}{lll} \mathrm{x} & = & \left[\begin{array}{lll} \mathrm{z} & = & e \\ \mathrm{w} & = & f \end{array}\right] \\ \mathrm{y} & = & \left[\begin{array}{lll} \mathrm{x} & = & g \\ \mathrm{y} & = & h \end{array}\right] \end{array}\right]$$

The following hold true:

$$\begin{array}{l} \mathrm{paths}(r_1) = \{\mathrm{x}, \mathrm{x.x}, \mathrm{x.y}, \mathrm{y}, \mathrm{y.x}, \mathrm{y.y}\} \\ \mathrm{paths}(r_2) = \{\mathrm{x}, \mathrm{x.z}, \mathrm{x.w}, \mathrm{y}, \mathrm{y.x}, \mathrm{y.y}\} \\ \mathrm{lcpaths}(\{r_1, r_2\}) = \{\mathrm{x}, \mathrm{y.x}, \mathrm{y.y}\} \end{array}$$

The notion of longest common paths for a set of records will be important when we discuss paths in record types below. [Is this still true?]

We now turn our attention to relabelling. An *L-relabelling* is a one-one function whose domain and range are included in L_+. We will standardly represent a relabelling, η, whose domain is $\{\pi_1, \ldots, \pi_n\}$ and which is defined by $\eta(\pi_1) = \pi'_1, \ldots, \eta(\pi_n) = \pi'_n$ as:

$$\begin{array}{l} \pi_1 \rightsquigarrow \pi'_1 \\ \vdots \\ \pi_n \rightsquigarrow \pi'_n \end{array}$$

If r is a record $\{\langle \ell_1, v_1\rangle, \ldots, \langle \ell_n, v_n\rangle\}$ and η is a relabelling, the *application of* η *to* r, $\eta * r$ is $\{\langle \ell'_1, v_1\rangle, \ldots, \langle \ell'_n, v_n\rangle\}$ where for each i such that $1 \leq i \leq n$, ℓ'_i is $\eta(\ell_i)$ if ℓ_i is in the domain of η and ℓ_i otherwise. Suppose η is the relabelling $\mathrm{x} \rightsquigarrow \mathrm{xx}, \mathrm{z} \rightsquigarrow \mathrm{zz}$ and r is as above. Then $\eta * r$ is

$$\left[\begin{array}{lll} \mathrm{xx} & = & \left[\begin{array}{lll} \mathrm{z} & = & a \\ \mathrm{w} & = & b \end{array}\right] \\ \mathrm{y} & = & c \end{array}\right]$$

That is, the label 'x' is changed, but nothing else.

We say that a relabelling, η, is *limited* to a record, r, just in case $\mathrm{dom}(\eta) \subseteq \mathrm{labels}(r)$.

We are, however, mainly interested in replacing total paths in a record which is why we have defined relabellings on the set of paths, L_+ rather than the set of labels, L. The *result of relabelling a record, r, with a relabelling, η*, in symbols, $[r]_\eta$, is $\varphi^-(\eta * \varphi(r))$. We say that η is a *proper relabelling* for r just in case $[r]_\eta$ is a record.

Suppose r is the record

$$\begin{bmatrix} \text{x} & = & \begin{bmatrix} \text{x} & = & a \\ \text{y} & = & b \end{bmatrix} \\ \text{y} & = & \begin{bmatrix} \text{x} & = & c \end{bmatrix} \end{bmatrix}$$

and η is the relabelling

$$\begin{array}{l} \text{x.x} \rightsquigarrow \text{y.z} \\ \text{x.y} \rightsquigarrow \text{x} \end{array}$$

$\varphi(r)$ is the flattened record

$$\begin{bmatrix} \text{x.x} & = & a \\ \text{x.y} & = & b \\ \text{y.x} & = & c \end{bmatrix}$$

$\eta * \varphi(r)$ is

$$\begin{bmatrix} \text{y.z} & = & a \\ \text{x} & = & b \\ \text{y.x} & = & c \end{bmatrix}$$

Then $[r]_\eta$ is the expansion of this record, that is, the record

$$\begin{bmatrix} \text{x} & = & b \\ \text{y} & = & \begin{bmatrix} \text{x} & = & c \\ \text{z} & = & a \end{bmatrix} \end{bmatrix}$$

and η is a proper relabelling for r.

Now let us consider a case where the relabelling is not proper. Suppose we start with the same record, r, as above but take a new relabelling, η':

$$\text{x.x} \rightsquigarrow \text{x}$$

Now $\eta' * \varphi(r)$ is the record

$$\begin{bmatrix} \text{x} & = & a \\ \text{x.y} & = & b \\ \text{y.x} & = & c \end{bmatrix}$$

However, the expansion $\varphi^{-}(\eta' * \varphi(r))$ is the set of ordered pairs:

$$\{\langle \text{x}, a\rangle, \langle \text{x}, \{\langle \text{y}, b\rangle\}\rangle, \langle \text{y}, \{\langle \text{x}, c\rangle\}\rangle\}$$

which is not a record since it is not the graph of a function: it has two ordered pairs whose first member is ‘x’. We can read the problem already from the flattened record since it has a label, ‘x’, which is a proper initial segment of another of its labels, ‘x.y’. Therefore η' is not a proper relabelling for r. Note that we cannot read the improperness off η' alone. η' is a proper relabelling for the record r':

$$\begin{bmatrix} \text{x} & = & \begin{bmatrix} \text{x} & = & a \end{bmatrix} \\ \text{y} & = & \begin{bmatrix} \text{x} & = & c \end{bmatrix} \end{bmatrix}$$

$[r']_{\eta'}$ is the record:

$$\begin{bmatrix} \text{x} & = & a \\ \text{y} & = & \begin{bmatrix} \text{x} & = & c \end{bmatrix} \end{bmatrix}$$

We use η^{-} to represent the inverse of the labelling η. That is, for any π, π', $\pi \rightsquigarrow \pi'$ is in η just in case $\pi' \rightsquigarrow \pi$ is in η^{-}.

Claim: If η is a proper relabelling for r limited to $\varphi(r)$, then η^- is a proper relabelling for $[r]_\eta$ and $[[r]_\eta]_{\eta^-} = r$.

The requirement that η is limited to $\varphi(r)$ is important here since it prevents cases where η^- would change labelling not changed by η. Consider an example where η is

$$\begin{array}{l} \mathrm{x} \rightsquigarrow \mathrm{z} \\ \mathrm{x.x} \rightsquigarrow \mathrm{y} \end{array}$$

and r is

$$\left[\begin{array}{lll} \mathrm{x} & = & \left[\begin{array}{lll} \mathrm{x} & = & a \end{array}\right] \\ \mathrm{z} & = & b \end{array}\right]$$

Note that in this case η is *not* limited to $\varphi(r)$ whose labels are 'x.x' and 'z' but do not include 'x'. η is proper for r and $[r]_\eta$ is

$$\left[\begin{array}{lll} \mathrm{y} & = & a \\ \mathrm{z} & = & b \end{array}\right]$$

Now consider η^- which is

$$\begin{array}{l} \mathrm{z} \rightsquigarrow \mathrm{x} \\ \mathrm{y} \rightsquigarrow \mathrm{x.x} \end{array}$$

The relabelling of $[r]_\eta$ by η^- will be the set of ordered pairs

$$\{\langle \mathrm{x}, b\rangle, \langle \mathrm{x}, \{\langle \mathrm{x}, a\rangle\}\rangle\}$$

which is not a record (since 'x' occurs twice as a label). Hence in this case η^- is not a proper relabelling for $[r]_\eta$ and $[[r]_\eta]_{\eta^-}$ is not identical with r. The reason for this is that η^- relabels a field that was not relabelled by η. Hence it is doing more than simply "undoing" the relabelling performed by η. This state of affairs will be prevented by requiring that η is limited to r, for example by removing 'x $\rightsquigarrow$ z' from η. Only paths changed by η will be changed back by η^- and our claim will hold true.

Using records as assignments

From Chapter 7, p. 384:

If r is a record, π is $\ell_0.\ell_1.\ldots.\ell_n$ where $\ell_0, \ell_1, \ldots, \ell_n$ are labels and v is an object of some type, then

> if $\pi \notin \mathrm{paths}(r)$, then $r[\pi = v]$ is the smallest record, r', such that
>
> 1. $\pi \in \mathrm{paths}(r')$
> 2. $r'.\pi = v$
> 3. for any $\pi' \in \mathrm{paths}(r)$, $\pi' \in \mathrm{paths}(r')$ and $r'.\pi' = r.\pi'$
>
> if $\pi \in \mathrm{paths}(r)$, then $r[\pi = v]$ is the smallest record, r', such that
>
> 1. $\pi \in \mathrm{paths}(r')$ and $r'.\pi = v$
> 2. for any $\pi' \in (\mathrm{paths}(r) - \{\pi'' \mid \pi'' \leq \pi\})$, $\pi' \in \mathrm{paths}(r')$ and $r'.\pi' = r.\pi'$

If r_1 is a record and r_2 is a record such $\mathrm{tpaths}(r_2) = \{\pi_1, \ldots, \pi_n\}$, then $r_1[r_2]$ is $r_1[\pi_1 = r_2.\pi_1] \ldots [\pi_n = r_2.\pi_n]$

11.2 Record types

We will model record types as flavoured labelled sets (see Appendix 1) using a distinguished flavour ‘RT’, the “record type flavour”. This is to distinguish record types from records, which may in certain cases correspond to the same set of ordered pairs.

A system of complex types **TYPE**$_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ *has (non-dependent) record types based on* $\langle \mathcal{L}, \mathbf{RType}\rangle$, where $\mathcal{L}$ is a countably infinite set (of labels) and **RType** $\subseteq$ **Type** (Chapter 1, p. 34) if

1. $Rec \in \mathbf{RType}$
2. $r :_{\mathbf{TYPE}_C} Rec$ iff r is a record according to $\mathcal{L}$ and **TYPE**$_C$.
3. $ERec \in \mathbf{RType}$
4. $r :_{\mathbf{TYPE}_C} ERec$ iff $r = \emptyset$
5. if $\ell \in \mathcal{L}$ and $T \in \mathbf{Type}$, then $\{\mathrm{RT}, \langle \ell, T\rangle\} \in \mathbf{RType}$.
6. $r :_{\mathbf{TYPE}_C} \{\mathrm{RT}, \langle \ell, T\rangle\}$ iff $r :_{\mathbf{TYPE}_C} Rec$, $\langle \ell, a\rangle \in r$ and $a :_{\mathbf{TYPE}_C} T$.

7. if $R \in \mathbf{RType} - \{Rec, ERec\}$, $\ell \in \mathcal{L}$, ℓ does not occur as a label in R (i.e. there is no field $\langle \ell', T' \rangle$ in R such that $\ell' = \ell$) and $T \in \mathbf{Type}$, then $R \cup \{\langle \ell, T \rangle\} \in \mathbf{RType}$.

8. $r :_{\mathbf{TYPE}_C} R \cup \{\langle \ell, T \rangle\}$ iff $r :_{\mathbf{TYPE}_C} R$, $\langle \ell, a \rangle \in r$ and $a :_{\mathbf{TYPE}_C} T$.

The notion of paths for record types is the set of paths viewing the record type as a labelled set but without those paths which include distinguished labels used in the labelled sets for non-record structures except in the case of those used in meet types: 'conj', 'conj_1' and 'conj_2'. We include modified versions of such paths where such labels have been removed. For example, the record type (assuming *Sit* is some basic type):

$$\{\mathrm{RT}, \langle \mathrm{e}, \{\langle \mathrm{conj}_1, \{\mathrm{RT}, \langle \mathrm{x}, \mathit{Ind} \rangle, \langle \mathrm{y}, \mathit{Ind} \rangle, \langle \mathrm{e}, \langle \lambda v_1{:}\mathit{Ind}, \lambda v_2{:}\mathit{Ind}.\mathrm{hug}(v_1, v_2), \langle \mathrm{x}, \mathrm{y} \rangle \rangle \rangle \} \rangle, \langle \mathrm{conj}_2, \mathit{Sit} \rangle \} \rangle \}$$

which we normally represent as

$$\left[\begin{array}{lll} \mathrm{e} & : & \left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{y} & : & \mathit{Ind} \\ \mathrm{e} & : & \mathrm{hug(x,y)} \end{array}\right] \wedge \mathit{Sit} \end{array}\right]$$

has the paths

$$\{\mathrm{e}, \mathrm{e.x}, \mathrm{e.y}, \mathrm{e.e}\}$$

This gives us non-dependent record types in a system of complex types. We can extend this to intensional systems of complex types (with stratification).

An *intensional system of complex types* $\mathbf{TYPE}_{IC}$ = $\langle$**Type**^{n}, **BType**, $\langle$**PType**^{n}, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F^n \rangle\rangle_{n \in Nat}$ *has (non-dependent) record types based on* $\langle \mathcal{L}, \mathbf{RType}^n \rangle_{n \in Nat}$ (Chapter 1, p. 37) if for each n, $\langle$**Type**^{n}, **BType**, $\langle$**PType**^{n}, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F^n \rangle\rangle$ has record types based on $\langle \mathcal{L}, \mathbf{RType}^n \rangle$ and

1. for each n, **RType**$^n \subseteq$ **RType**$^{n+1}$

2. for each $n > 0$, *RecType*$^n \in$ **Type**^{n}

3. for each $n > 0$, $T :_{\mathbf{TYPE}_{IC_n}} \mathit{RecType}^n$ iff $T \in \mathbf{RType}^{n-1}$

Here, but not in Cooper (2012b), we make explicit that *RecType* is treated in a similar manner to *Type*, that is, it is a distinguished urelement and *RecType*^{n} represents the labelled set $\{\langle \mathrm{ord}, n\rangle, \langle \mathrm{typ}, \mathit{RecType}\rangle\}$ where 'ord' and 'typ' are reserved labels ("order", "type").

Intensional type systems may in addition contain *dependent* record types.

An *intensional system of complex types* **TYPE**$_{IC}$ = $\langle$**Type**^{n}, **BType**, $\langle$**PType**^{n}, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F^n\rangle\rangle_{n\in Nat}$ *has dependent record types based on* $\langle \mathcal{L}, \mathbf{RType}^n\rangle_{n\in Nat}$, (Chapter 1, p. 38) if it has record types based on $\langle \mathcal{L}, \mathbf{RType}^n\rangle_{n\in Nat}$ and for each $n > 0$

1. if $R \in$ **RType**^{n}, $\ell \in \mathcal{L} - \mathrm{labels}(R)$, $T_1, \ldots, T_m \in \mathbf{Type}^n$, $\pi_1, \ldots, \pi_m \in \mathrm{paths}(R)$ and $\mathcal{F}$ is a function of type $(T_1 \rightarrow \ldots \rightarrow (T_m \rightarrow \mathit{Type}^n)\ldots)$, then $R \cup \{\langle \ell, \langle \mathcal{F}, \langle \pi_1, \ldots, \pi_m\rangle\rangle\rangle\} \in \mathbf{RType}^n$.
2. $r :_{\mathbf{TYPE}_{IC_n}} R \cup \{\langle \ell, \langle \mathcal{F}, \langle \pi_1, \ldots, \pi_m\rangle\rangle\rangle\}$ iff $r :_{\mathbf{TYPE}_{IC_n}} R$, $\langle \ell, a\rangle$ is a field in r, $r.\pi_1 :_{\mathbf{TYPE}_{IC_n}} T_1, \ldots, r.\pi_m :_{\mathbf{TYPE}_{IC_n}} T_m$ and $a :_{\mathbf{TYPE}_{IC_n}} \mathcal{F}(r.\pi_1)\ldots(r.\pi_m)$.

We represent a record type $\{\mathrm{RT}, \langle \ell_1, T_1\rangle, \ldots, \langle \ell_n, T_n\rangle\}$ graphically as

$$\begin{bmatrix} \ell_1 & : & T_1 \\ \ldots & & \\ \ell_n & : & T_n \end{bmatrix}$$

In the case of a manifest field, that is, one containing a singleton type, as in $\langle \ell, T_a\rangle$, we display this

$$\begin{bmatrix} \ell{=}a & : & T \end{bmatrix}$$

In the case of a multiple singleton type (a singleton type formed from a singleton type) as in $\langle \ell, T_{a,b,\ldots}\rangle$, we display

$$\begin{bmatrix} \ell{=}a, b, \ldots & : & T \end{bmatrix}$$

In the case of dependent record types we sometimes use a convenient notation representing e.g.

$$\langle \lambda u \lambda v\ \mathrm{love}(u, v), \langle \pi_1, \pi_2\rangle\rangle$$

as

$$\text{love}(\pi_1, \pi_2)$$

Our systems now allow both function types and dependent record types and allow dependent record types to be arguments to functions. We have to be careful when considering what the result of applying a function to a dependent record type should be. Consider the following simple example:

$$\lambda v_0 : \textit{RecType} \,.\, \left[\mathrm{c}_0 : v_0\right]$$

What should be the result of applying this function to the record type

$$\begin{bmatrix} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_1 & : & \langle \lambda v_1 : \textit{Ind} \,.\, \mathrm{dog}(v_1), \langle \mathrm{x} \rangle \rangle \end{bmatrix}$$

Given normal assumptions about function application the result would be

$$\begin{bmatrix} \mathrm{c}_0 & : & \begin{bmatrix} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_1 & : & \langle \lambda v_1 : \textit{Ind} \,.\, \mathrm{dog}(v_1), \langle \mathrm{x} \rangle \rangle \end{bmatrix} \end{bmatrix} \text{(incorrect!)}$$

but this would be incorrect. In fact it is not a well-formed record type since x is not a path in it. Instead the result should be

$$\begin{bmatrix} \mathrm{c}_0 & : & \begin{bmatrix} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_1 & : & \langle \lambda v_1 : \textit{Ind} \,.\, \mathrm{dog}(v_1), \langle \mathrm{c}_0.\mathrm{x} \rangle \rangle \end{bmatrix} \end{bmatrix}$$

where the path from the top of the record type is specified. However, in the abbreviatory notation we write just ‘x’ when the label is used as an argument and interpret this as the path from the top of the record type to the field labelled ‘x’ in the local record type. Thus we will write

$$\begin{bmatrix} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_1 & : & \mathrm{dog}(\mathrm{x}) \end{bmatrix}$$

(where the ‘x’ in ‘dog(x)’ signifies the path consisting of just the single label ‘x’) and

$$\left[\begin{array}{lll} \mathrm{c}_0 & : & \left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_1 & : & \mathrm{dog(x)} \end{array}\right] \end{array}\right]$$

(where the ‘x’ in ‘dog(x)’ signifies the path from the top of the record type down to ‘x’ in the local record type, that is, ‘c_0.x’).[4]

Note that this adjustment of paths is only required when a record type is being substituted into a position that lies on a path within a resulting record type. It will not, for example, apply in a case where a record type is to be substituted for an argument to a predicate such as when applying the function

$$\lambda v_0 : \textit{RecType} \,.\, \left[\mathrm{c}_0\mathrm{:appear}(v_0)\right]$$

to

$$\left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_1 & : & \langle \lambda v : \textit{Ind} \,.\, \mathrm{dog}(v),\, \langle \mathrm{x} \rangle \rangle \\ \mathrm{c}_2 & : & \langle \lambda v : \textit{Ind} \,.\, \mathrm{approach}(v),\, \langle \mathrm{x} \rangle \rangle \end{array}\right]$$

where the position of v_0 is in an “intensional context”, that is, as the argument to a predicate and there is no path to this position in the record type resulting from applying the function. Here the result of the application is

$$\left[\begin{array}{lll} \mathrm{c}_0 & : & \mathrm{appear(} \left[\begin{array}{lll} \mathrm{x} & : & \textit{Ind} \\ \mathrm{c}_1 & : & \langle \lambda v : \textit{Ind} \,.\, \mathrm{dog}(v),\, \langle \mathrm{x} \rangle \rangle \\ \mathrm{c}_2 & : & \langle \lambda v : \textit{Ind} \,.\, \mathrm{approach}(v),\, \langle \mathrm{x} \rangle \rangle \end{array}\right] \mathrm{)} \end{array}\right]$$

with no adjustment necessary to the paths representing the dependencies.[5] (Note that ‘c_0.x’ is not a path in this record type.)

Suppose that we wish to represent a type which requires that there is some dog such that it appears to be approaching (that is a *de re* reading). In the abbreviatory notation we might be tempted to write

[4]This convention of representing the path from the top of the record type to the “local” field by the final label on the path is new since Cooper (2012b).

[5]This record corresponds to the interpretation of *it appears that a dog is approaching*.

$$\left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c}_1 & : & \mathrm{dog(x)} \\ \mathrm{c}_0 & : & \mathrm{appear}(\left[\begin{array}{lll} \mathrm{c}_2 & : & \mathrm{approach(x)} \end{array}\right]) \end{array}\right] \text{(incorrect!)}$$

corresponding to

$$\left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c}_1 & : & \langle \lambda v{:}\mathit{Ind}\ .\ \mathrm{dog}(v), \langle \mathrm{x} \rangle\rangle \\ \mathrm{c}_0 & : & \mathrm{appear}(\left[\begin{array}{lll} \mathrm{c}_2 & : & \langle \lambda v{:}\mathit{Ind}\ .\ \mathrm{approach}(v), \langle \mathrm{x} \rangle\rangle \end{array}\right]) \end{array}\right] \text{(incorrect!)}$$

This is, however, incorrect since it refers to a path 'x' in the type which is the argument to 'appear' which does not exist. Instead we need to refer to the path 'x' in the record type containing the field labelled 'c_0':

$$\left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c}_1 & : & \langle \lambda v{:}\mathit{Ind}\ .\ \mathrm{dog}(v), \langle \mathrm{x} \rangle\rangle \\ \mathrm{c}_0 & : & \langle \lambda v{:}\mathit{Ind}\ .\ \mathrm{appear}(\left[\begin{array}{lll} \mathrm{c}_2 & : & \mathrm{approach}(v) \end{array}\right]), \langle \mathrm{x} \rangle\rangle \end{array}\right]$$

In the abbreviatory notation we will use '$\Uparrow$' to indicate that the path referred to is in the "next higher" record type[6]:

$$\left[\begin{array}{lll} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c}_1 & : & \mathrm{dog(x)} \\ \mathrm{c}_0 & : & \mathrm{appear}(\left[\begin{array}{lll} \mathrm{c}_2 & : & \mathrm{approach}(\Uparrow\mathrm{x}) \end{array}\right]) \end{array}\right]$$

These matters arise as a result of our choice of using paths to represent dependencies in record types (rather than, for example, introducing additional unique identifiers to keep track of the positions within a record type as has been suggested by Thierry Coquand). It seems like a matter of implementation rather than a matter of substance and it is straightforward to define a path-aware notion of substitution which can be used in the definition of what it means to apply a TTR function to an argument. If f is a function represented by $\lambda v : T\ .\ \varphi$ and α is the representation of an object of type T, then the result of applying f to α, $f(\alpha)$, is represented by Subst(α,v,φ,$\emptyset$), that is, the result of substituting α for v in φ with respect to the empty path where for arbitrary α, v, φ, π, Subst(α,v,φ,π) is defined as

1. extend-paths(α,π), if φ is v

[6]This notation is new since Cooper (2012b).

2. φ, if φ is of the form $\lambda v : T . \zeta$, for some T and ζ (i.e. don't do any substitution if v is bound within φ)

3. $\lambda u : T$. Subst(α,v,ζ,π), if φ is of the form $\lambda u : T.\zeta$ and u is not v.

4. $\left[\begin{array}{lll} \ell_1 & : & \text{Subst}(\alpha,v,T_1,\pi.\ell_1) \\ \ldots & & \\ \ell_n & : & \text{Subst}(\alpha,v,T_n,\pi.\ell_n) \end{array}\right]$, if φ is $\left[\begin{array}{lll} \ell_1 & : & T_1 \\ \ldots & & \\ \ell_n & : & T_n \end{array}\right]$

5. P(Subst(α,v,β_1,π),...,Subst(α,v,β_n,π)), if α is $P(\beta_1, \ldots, \beta_n)$ for some predicate P

6. φ otherwise

extend-paths(α,π) is

1. $\langle f, \langle \pi.\pi_1, \ldots, \pi.\pi_n \rangle\rangle$, if α is $\langle f, \langle \pi_1, \ldots, \pi_n \rangle\rangle$

2. $\left[\begin{array}{lll} \ell_1 & : & \text{extend-paths}(T_1, \pi) \\ \ldots & & \\ \ell_n & : & \text{extend-paths}(T_n, \pi) \end{array}\right]$ if α is $\left[\begin{array}{lll} \ell_1 & : & T_1 \\ \ldots & & \\ \ell_n & : & T_n \end{array}\right]$

3. P(extend-paths(β_1, π),...,extend-paths(β_n, π)), if α is $P(\beta_1, \ldots, \beta_n)$ for some predicate P

4. α, otherwise

12 Merges of record types

This definition of the merge operation on types is first introduced and discussed in Chapter 2, p. 88. We define a function μ which maps meets of record types to an equivalent record type, record types to equivalent types where meets in their values have been simplified by μ and any other types to themselves. μ is also defined on labelled sets which are not types in order to account for merging inside a record type of structures which depend on fields in the type outside the structure:

1. if for some $T_1, T_2, T = (T_1 \wedge T_2)$ and $T_1 \sqsubseteq T_2$ then $\mu(T) = T_1$

2. if for some $T_1, T_2, T = (T_1 \wedge T_2)$ and $T_2 \sqsubseteq T_1$ then $\mu(T) = T_2$

3. otherwise:

   a) if for some labelled sets $T_1, T_2, T = (T_1 \wedge T_2)$ then $\mu(T) = \mu'(\mu(T_1) \wedge \mu(T_2))$.

   b) if T is a labelled set then $\mu(T)$ is T' such that for any ℓ,v, $\langle \ell, \mu(v) \rangle \in T'$ iff $\langle \ell, v \rangle \in T$.

   c) otherwise $\mu(T) = T$.

$\mu'(T_1 \wedge T_2)$ is defined by:

1. if T_1 and T_2 are labelled sets, then $\mu'(T_1 \wedge T_2) = T_3$ such that
   a) for any ℓ, v_1, v_2, if $\langle \ell, v_1 \rangle \in T_1$ and $\langle \ell, v_2 \rangle \in T_2$, then
      i. if v_1 and v_2 are
         $\langle \lambda u_1{:}T'_1 \ldots \lambda u_i{:}T'_i \, . \, \phi, \langle \pi_1 \ldots \pi_i \rangle \rangle$
         and
         $\langle \lambda u'_1{:}T''_1 \ldots \lambda u'_k{:}T''_k \, . \, \psi, \langle \pi'_1 \ldots \pi'_k \rangle \rangle$
         respectively, then
         $\langle \lambda u_1{:}T'_1 \ldots \lambda u_i{:}T'_i, \lambda u'_1{:}T''_1 \ldots \lambda u'_k{:}T''_k \, . \, \mu(\phi \wedge \psi), \langle \pi_1 \ldots \pi_i, \pi'_1 \ldots \pi'_k \rangle \rangle \in T_3$
      ii. if v_1 is
         $\langle \lambda u_1{:}T'_1 \ldots \lambda u_i{:}T'_i \, . \, \phi, \langle \pi_1 \ldots \pi_i \rangle \rangle$
         and v_2 is a type (i.e. not of the form $\langle f, \Pi \rangle$ for some function f and sequence of paths Π), then
         $\langle \lambda u_1{:}T'_1 \ldots \lambda u_i{:}T'_i \, . \, \mu(\phi \wedge v_2), \langle \pi_1 \ldots \pi_i \rangle \rangle \in T_3$
      iii. if v_2 is
         $\langle \lambda u'_1{:}T''_1 \ldots \lambda u'_k{:}T''_k \, . \, \psi), \langle \pi'_1 \ldots \pi'_k \rangle \rangle$
         and v_1 is a type, then
         $\langle \lambda u'_1{:}T''_1 \ldots \lambda u'_k{:}T''_k \, . \, \mu(v_1 \wedge \psi)), \langle \pi'_1 \ldots \pi'_k \rangle \rangle \in T_3$
      iv. otherwise $\langle \ell, \mu(v_1 \wedge v_2) \rangle \in T_3$
   b) for any ℓ, v_1, if $\langle \ell, v_1 \rangle \in T_1$ and there is no v_2 such that $\langle \ell, v_2 \rangle \in T_2$, then $\langle \ell, v_1 \rangle \in T_3$
   c) for any ℓ, v_2, if $\langle \ell, v_2 \rangle \in T_2$ and there is no v_1 such that $\langle \ell, v_1 \rangle \in T_1$, then $\langle \ell, v_2 \rangle \in T_3$
2. if T_1 is $\text{list}(T'_1)$ ($\text{set}(T'_1)$, $\text{plurality}(T'_1)$) and T_2 is $\text{list}(T'_2)$ ($\text{set}(T'_2)$, $\text{plurality}(T'_2)$), then $\mu'(T_1 \wedge T_2) = \text{list}(\mu(T'_1 \wedge T'_2))$ ($\text{set}(\mu(T'_1 \wedge T'_2))$, $\text{plurality}(\mu(T'_1 \wedge T'_2))$)
3. otherwise $\mu'(T_1 \wedge T_2) = T_1 \wedge T_2$

$(T_1 \,\dot{\wedge}\, T_2)$ is used to represent $\mu(T_1 \wedge T_2)$. We call $(T_1 \,\dot{\wedge}\, T_2)$ the *merge* of T_1 and T_2.

We define also a notion of *asymmetric merge* (Chapter 2, p. 90) of T_1 and T_2 which is defined by a function, μ_{asym}, exactly like μ except that the first two clauses of the definition of μ are missing and μ' is replaced by another function μ'_{asym}. Thus the definition of μ_{asym} is:

1. if for some $T_1, T_2, T = (T_1 \wedge T_2)$ then $\mu_{\text{asym}}(T) = \mu'_{\text{asym}}(\mu_{\text{asym}}(T_1) \wedge \mu_{\text{asym}}(T_2))$.
2. if T is a record type then $\mu_{\text{asym}}(T)$ is T' such that for any ℓ,v, $\langle \ell, \mu_{\text{asym}}(v) \rangle \in T'$ iff $\langle \ell, v \rangle \in T$.

3. otherwise $\mu_{\mathrm{asym}}(T) = T$.

The definition of μ'_{asym} is exactly like μ', replacing μ and μ' with μ_{asym} and μ'_{asym} respectively, except that the clause 3 of the definition of μ' is replaced by

3′. otherwise $\mu'_{\mathrm{asym}}(T_1 \wedge T_2) = T_2$

We use $T_1 \boxed{\underset{\cdot}{\wedge}}\, T_2$ to represent the asymmetric merge of T_1 and T_2.

Asymmetric merge may result in an ill-formed record type if we take the asymmetric merge of a record type, T_1, and a non-record type, T_2, since T_1 may be embedded in a larger type with fields dependent on paths into T_1 which will not be present in the result where T_2 has been substituted for T_1 thus removing the relevant paths.

Merging functions which return types $\lambda r : T_1 \;.\; T_2(r) \underset{:}{\wedge} \lambda r : T_3 \;.\; T_4(r)$ denotes the function $\lambda r : T_1 \underset{\cdot}{\wedge} T_3 \;.\; T_2(r) \underset{\cdot}{\wedge} T_4(r)$.

Constructing fixed point types for functions which return types (Chapter 5, p. 240ff.)

If T is a record type and r is a record of any type, we will use $T_{r.\pi \rightsquigarrow \pi}$ to designate the type like T except that for any $\pi \in \mathrm{paths}(T)$ any occurrence of $r.\pi$ is replaced by π.

A dependent type, $\mathcal{T}$, is *path-dependent on* r, $\mathcal{T}(\!(r)\!)_{\text{path}}$, just in case $\mathcal{T}$ depends on paths in r but not on the whole object r.

If $\mathcal{T}$ is a dependent record type of the form $\lambda r : T_1 \;.\; T_2(\!(r)\!)_{\text{path}}$ where T_1 is a record type and for any r, $\mathrm{paths}(T_1) \cap \mathrm{paths}(\mathcal{T}(r)) = \emptyset$, then

$\mathcal{F}(\mathcal{T})$ is that type T such that for any $r^* : T_1$, $\lambda r : T_1 \;.\; (T_1 \underset{\cdot}{\wedge} T_2)_{r.\pi \rightsquigarrow \pi}(r^*) = T$

If $\mathcal{T}$ is a dependent record type of the form $\lambda r : T_1 \;.\; T_2(\!(r)\!)$ where T_1 is a record type, then

$\mathcal{F}_{\text{quasi}}(\mathcal{T})$ is that type T such that for any $r^* : T_1$, $\lambda r : T_1 \;.\; (\big[\mathfrak{c}^* {:} T_1\big] \underset{\cdot}{\wedge} \big[\mathfrak{c}^* {:} T_2\big])_{r.\pi \rightsquigarrow \mathfrak{c}^*.\pi}(r^*) = T$

If $\mathcal{T}$ is a dependent record type of the form $\lambda r : T_1 \;.\; T_2(\!(r)\!)$ where T_1 is a record type and for any r in its domain $\mathcal{T}(r)$ is a record type, then

$$\mathcal{F}_{\text{quasi}^*}(\mathcal{T}) = \mathcal{F}_{\text{quasi}}((T))$$

else if $\mathcal{T}$ is a dependent record type of the form $\lambda r : T_1 \,.\, \mathcal{T}'(\!(r)\!)$ where T_1 is a record type and for any r in its domain $\mathcal{T}(r)$ is a dependent record type, then

$$\mathcal{F}_{\text{quasi}^*}(\mathcal{T}) \text{ is that type } T \text{ such that for any } r^* : T_1, T = \lambda r : T_1 \,.\, (\left[\mathfrak{c}^* : T_1\right] \dot{\wedge} \left[\mathfrak{r}^* : \mathcal{F}_{\text{quasi}^*}(\mathcal{T}')\right])_{r.\pi \rightsquigarrow \mathfrak{c}^*.\pi}$$

13 Unique identifier notation for record types

The unique identifier notation for a type, T, can be obtained from the official notation for T by carrying out in order the following manipulations (Chapter 2, p. 79):

1. Add a unique identifier $\boxed{i}$ (where i is a natural number) to the last label occurrence in any path, π, which is referenced in a dependent field, $\langle f, \langle \ldots, \pi, \ldots \rangle \rangle$, somewhere in T.
2. Replace π in any dependent field, $\langle f, \langle \ldots, \pi, \ldots \rangle \rangle$, with the unique identifier associated with the final label occurrence in π.
3. Replace any pair in a dependent field of the form

   $$\langle \lambda v_1 : T_1 \ldots \lambda v_n : T_n \,.\, \varphi(\!(v_1, \ldots, v_n)\!), \langle \boxed{i_1}, \ldots, \boxed{i_n} \rangle \rangle$$

   with

   $$\varphi(\!(\boxed{i_1} : T_1, \ldots, \boxed{i_n} : T_n)\!)$$

We also show how to convert back from unique identifier notation for a type, T, to official notation. We can do this by carrying out the following manipulations (Chapter 2, p. 80):

1. For each occurrence of $\boxed{i}$ in a dependent field labelled ℓ_d (the label for the dependent field) for some natural number, i, locate the smallest record structure, Δ, in T which contains a path, π_d, to ℓ_d and a path π_a to a label $\ell_{\boxed{i}}$ (the label for the addressed field)
2. Let ℓ be the first label on π_d. If $\Delta.\ell$ is an ordered pair $\langle \lambda v_j : T_j \,.\, \varphi(\!(\boxed{i} : T_i)\!), \langle \pi_1, \ldots, \pi_n \rangle \rangle$, then replace $\Delta.\ell$ with $\langle \lambda v_i : T_i \,.\, \lambda v_j : T_j \,.\, \varphi[\boxed{i} : T_i \rightsquigarrow v_i], \langle \pi_1, \ldots, \pi_n, \pi_a \rangle \rangle$. Otherwise, if $\Delta.\ell$ is $\varphi(\!(\boxed{i} : T_i)\!)$, replace $\Delta.\ell$ with $\langle \lambda v_i : T_i \,.\, \varphi[\boxed{i} : T_i \rightsquigarrow v_i], \langle \pi_a \rangle \rangle$
3. For any number i, remove the subscript $\boxed{i}$ on any label on which it occurs.

14 Dependency in record types and record type generalization

If T is a record type, $\pi_1 \in \text{paths}(T)$ and $\pi_2 \in \text{tpaths}(T)$, then π_2 *depends on* π_1 (Chapter 7, p. 387) iff, in unique identifier notation, π_1 is indexed with $\boxed{i}$ and the representation of $T.\pi_2$ contains $\boxed{i}$, for some natural number i.

If T is a record type and $\pi \in \text{paths}(T)$ then *the dependency family of* π *in* T (Chapter 7, p. 387), $\text{paths}_\pi(T)$, is that subset, Π, of $\text{paths}(T)$ such that

1. $\pi \in \Pi$
2. for any $\pi' \in \Pi$ and $\pi'' \in \text{tpaths}(T)$, if π'' depends on π', then $\pi'' \in \Pi$
3. for and $\pi' \in \Pi$ and $\pi'' \in \text{paths}(T)$, if π' depends on π'', then $\pi'' \in \Pi$

If T is a record type and $\pi \in \text{paths}(T)$, then T *generalized to* π (Chapter 7, p. 387), T^π, is the smallest labelled set T' such that $\text{paths}_\pi(T) \subseteq \text{paths}(T')$ and for all $\pi' \in \text{tpaths}(T')$, $T'.\pi = T.\pi$.

If T is a record type and $\{\pi_1, \ldots, \pi_n\} \subset \text{paths}(T)$, then T *generalized to* $\pi_1, \ldots, \pi_n$, $T^{\pi_1,\ldots,\pi_n}$ (Chapter 7, p. 388), is the smallest labelled set T' such that for all $\pi_i \in \{\pi_1, \ldots, \pi_n\}$, $\text{paths}_{\pi_i}(T) \subseteq \text{paths}(T')$ and for all $\pi' \in \text{tpaths}(T')$, $T'.\pi = T.\pi$.

15 Using records to restrict record types

(These definitions were not included in Cooper, 2012b.)

A type system $\mathbb{T}$ *has restricted types according to a set of labels* $\mathcal{L}$ (Chapter 3, p. 128) if it is the case that

1. if T is a type, but not a record type, according to $\mathbb{T}$ and r is a record according to $\mathcal{L}$ and $\mathbb{T}$, then the restriction of T by r, $\rho(T, r)$, is a type according to $\mathbb{T}$.
2. $a :_{\mathbb{T}} \rho(T, r)$ iff $a \underline{\varepsilon} r$ and $a :_{\mathbb{T}} T$.

We then generalize restriction to record types and other objects using $o \restriction r$ to represent the object o restricted to r.

1. If T is a type but not a record type then

$$T \restriction r = \rho(T, r)$$

2. If X is a labelled set whose labels are not distinguished (as for example in ptypes, that is $\text{labels}(X)$ is a set of labels which can be used in record types)

$$\{\langle \ell_1, o_1 \rangle, \ldots, \langle \ell_n, o_n \rangle\}$$

then

$$X \restriction r = \{\langle \ell_1, o_1 \restriction r \rangle, \ldots, \langle \ell_n, o_n \restriction r \rangle\}$$

3. if o is

$$\langle \mathcal{T}, \Pi \rangle$$

where $\mathcal{T}$ is a dependent type and Π is a sequence of paths, then

$$o \restriction r = \langle \mathcal{T} \restriction r, \Pi \rangle$$

4. if $\mathcal{T}$ is a dependent type

$$\lambda v_1 : T_1 \ . \ \ldots \ . \ \lambda v_n : T_n \ . \ T(\!(v_1, \ldots, v_n)\!)$$

then

$$\mathcal{T} \restriction r = \lambda v_1 : T_1 \ . \ \ldots \ . \ \lambda v_n \ . \ T(\!(v_1, \ldots, v_n)\!) \restriction r$$

5. otherwise $o \restriction r = o$

We use a similar notation for restricted fields in record types as we do for manifest fields. That is, we represent (1a) as (1b) in the case where T is not a record type or a pair of a dependent type and a sequence of paths.

(1) a. $\left[\ \ell \ : \ T \restriction r\ \right]$
b. $\left[\ \ell_{\underline{\varepsilon}} r \ : \ T\ \right]$

If T is a record type and r is a record, then $T \| \restriction r$, the *restriction* of T by r is the result of replacing each field, $\langle \ell, T' \rangle$, in T such that ℓ is a label in r, with

The remaining versions of restriction are perhaps not necessary

1. $\langle \ell, T' \restriction r.\ell \rangle$[7], if T' is a type
2. $\langle \ell, \langle f', \Pi \rangle \rangle$, if $T' = \langle f, \Pi \rangle$ where f is a function and Π is a sequence of paths of length n and for any $a_1, \ldots, a_n$, $f'(a_1) \ldots (a_n)$ is defined iff $f(a_1) \ldots (a_n)$ is defined and $f'(a_1) \ldots (a_n) = f(a_1) \ldots (a_n) \restriction r.\ell$

[7]That is, in an adaptation of our graphical notation for manifest fields, $\left[\ell{:}T'\right]$ is replaced by $\left[\ell_{\underline{\varepsilon}} r.\ell{:}T'\right]$

A variant of this notion of restriction is default restriction which will only require restriction of fields which are not already restricted. If T is a record type and r is a record, then $T /\!\!/\!\!\upharpoonright r$, the *default restriction* of T by r is the result of replacing each field, $\langle \ell, T' \rangle$, in T such that ℓ is a label in r, with

1. $\langle \ell, T' \upharpoonright r.\ell \rangle$, if T' is a type but not a restricted type. If T' is a singleton type then $\langle \ell, T' \rangle$ is replaced by itself.

2. $\langle \ell, \langle f', \Pi \rangle \rangle$, if $T' = \langle f, \Pi \rangle$ where f is a function and Π is a sequence of paths of length n and for any $a_1, \ldots, a_n$, $f'(a_1) \ldots (a_n)$ is defined iff $f(a_1) \ldots (a_n)$ is defined and $f'(a_1) \ldots (a_n) = f(a_1) \ldots (a_n) \upharpoonright r.\ell$ if $f(a_1) \ldots (a_n)$ is not a restricted type. Otherwise, $f'(a_1) \ldots (a_n) = f(a_1) \ldots (a_n)$.

16 Generalizing record types

If T : *RecType*, $\ell \in$ labels(T) and $\langle \ell, T' \rangle \in T$ where T': *Type* (that is, $[\ell, T']$ is a non-dependent field in T), then *the generalization of T to its ℓ-field* (Chapter 7, p. 354, T^{ℓ}, is

$$\left[\begin{array}{lcl} \ell & : & T' \end{array}\right]$$

17 Path alignment within record types

(Chapter 8, p. 430) Suppose that T is a record type and that π_1 and π_2 are paths in T. Then we use $T_{\pi_1=\pi_2}$ to represent the type exactly like T except that $T_{\pi_1=\pi_2}.\pi_1 = (T.\pi_1)_{\pi_2}$, that is, whatever type, T', is at the end of the path π_1, is replaced by the singleton type T'_{π_2}, or if $T.\pi_1$ is

$$\langle \lambda v_1 \!:\! T_1 \ldots \lambda v_n \!:\! T_n \,.\, T'(\!(v_1, \ldots, v_n)\!), \Pi \rangle$$

then it is replaced by

$$\langle \lambda v_1 \!:\! T_1 \ldots \lambda v_n \!:\! T_n \,.\, (T'(\!(v_1, \ldots, v_n)\!))_{\pi_2}, \Pi \rangle$$

We use $T_{\pi_{11}=\pi_{21},\ldots,\pi_{1n}=\pi_{2n}}$ to represent $(\ldots(T_{\pi_{11}=\pi_{21}})\ldots)_{\pi_{1n}=\pi_{2n}}$.

(Chapter 8, p. 434.) If $\varphi = \ulcorner \lambda r \!:\! T \,.\, \psi \urcorner$ and $\pi_1, \pi_2 \in \mathrm{paths}(T)$, then

$$\varphi_{\pi_1=\pi_2} = \ulcorner \lambda r \!:\! T_{\pi_1=\pi_2} \,.\, \psi \urcorner$$

18 List types

List types were not included in Cooper (2012b).

A system of complex types with record types $\mathbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle$⟩ *has list types* (Chapter 2, p. 58) if

1. for any $T \in$ **Type**, $\mathrm{list}(T) \in$ **Type**
2. for any $T \in$ **Type**,
   a) $[\,] :_{\mathbf{TYPE_C}} \mathrm{list}(T)$
   b) $a\,|\,L :_{\mathbf{TYPE_C}} \mathrm{list}(T)$ iff $a :_{\mathbf{TYPE_C}} T$ and $L :_{\mathbf{TYPE_C}} \mathrm{list}(T)$

In Cooper (2012b) we did not specify an encoding of lists in terms of sets. Here we will use records with the reserved labels 'fst' and 'rst' for the first member of the list and the remainder ("rest") of the list respectively. We let the empty list, [], be the empty set, ∅.[8] If L is a list then $a\,|\,L$ is to be the record

$$\begin{bmatrix} \text{fst} & = & a \\ \text{rst} & = & L \end{bmatrix}$$

We sometimes use $\mathrm{nelist}(T)$ as an abbreviation for the type of non-empty lists:

$$\begin{bmatrix} \text{fst} & : & T \\ \text{rst} & : & \mathrm{list}(T) \end{bmatrix}$$

If L is a list we often use $\mathrm{fst}(L)$ and $\mathrm{rst}(L)$ to represent L.fst and L.rst respectively.

In contrast to Cooper (2012b) we here make it explcit that $\mathrm{list}(T)$ represents $\{\langle \mathrm{lst}, T\rangle\}$ where 'lst' is a reserved label.

19 Strings and regular types

A system of complex types $\mathbf{TYPE}_C$ = ⟨**Type**, **BType**, ⟨**PType**, **Pred**, **ArgIndices**, *Arity*⟩, $\langle A, F\rangle$⟩ with record types based on $\langle \mathcal{L}, \mathbf{RType}\rangle$ *has strings* (Chapter 2, p. 45) if

[8] If it is important to distinguish the empty list from the empty set we could use an additional reserved label, e.g. 'lst', and have the empty list be the labelled set $\{\langle \mathrm{lst}, \emptyset\rangle\}$.

1. for each natural number i, $\mathrm{t}_i \in \mathcal{L}$
2. *String* $\in$ **BType**
3. $\emptyset :_{\mathbf{TYPE}_C}$ *String*
4. if $T \in$ **Type** and $a :_{\mathbf{TYPE}_C} T$ then $\{\langle \mathrm{t}_0, a\rangle\}$: *String*
5. if $s :_{\mathbf{TYPE}_C}$ *String*, $\mathrm{t}_n \in \mathrm{labels}(s)$ such that there is no $i > n$ where $\mathrm{t}_i \in \mathrm{labels}(s)$, $T \in$ **Type** and $a :_{\mathbf{TYPE}_C} T$ then $s \cup \{\langle \mathrm{t}_{n+1}, a\rangle\} :_{\mathbf{TYPE}_C}$ *String*
6. Nothing is of type *String* except as required above.

If s is a string according to some type system with strings we write $\mathrm{length}(s)$ for $|s|$ (that is the cardinality of the set of ordered pairs constituting the record modelling the string). If s_1 and s_2 are strings, then the concatenation $s_1 s_2$ is $s_1 \cup s_2'$ where s_2' is the result of replacing each label t_i in $\mathrm{labels}(s_2)$ with $t_{i+\mathrm{length}(s_1)}$.

If s is a string according to some type system and $\mathrm{t}_n \in \mathrm{labels}(s)$, we use $s[n]$ to represent $s.\mathrm{t}_n$. We use ε to represent the empty string (which is identical with the empty set).

A system of complex types $\mathbf{TYPE}_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ with strings has *length determining string types* (Chapter 2, p. 47) if

1. for any $T \in$ **Type** and n a natural number, the string types $T^{=n}$, $T^{\leq n}$, $T^{\geq n} \in$ **Type**
2. $s :_{\mathbf{TYPE}_C} T^{=n}$ $(T^{\leq n}, T^{\geq n})$ iff $s :_{\mathbf{TYPE}_C}$ *String*, for all i, $0 \leq i < \mathrm{length}(s)$, $s[i] :_{\mathbf{TYPE_C}} T$ and $\mathrm{length}(s) = (\leq, \geq)\ n$

When there is not source of confusion we write T^n for $T^{=n}$. We also write T^* for $T^{\geq 0}$ (Kleene star) and T^+ for $T^{\geq 1}$ (Kleene plus).

As with other complex types we model $T^{\xi n}$ (where ξ is '$=$', '$\leq$' or '$\geq$') as a labelled set:

$$\{\langle \mathrm{str}_0, \{\langle \mathrm{comp}, \xi\rangle, \langle \mathrm{num}, n\rangle, \langle \mathrm{type}, T\rangle\}\rangle\}$$

A system of complex types $\mathbf{TYPE}_C$ = $\langle$**Type**, **BType**, $\langle$**PType**, **Pred**, **ArgIndices**, *Arity*$\rangle$, $\langle A, F\rangle\rangle$ with strings and length determining string types has *concatenation types* (Chapter 2, p. 48) if

1. if $T_1, T_2 \in$ **Type** then the string type $T_1{}^\frown T_2 \in$ **Type**

2. $s :_{\mathbf{TYPE}_C} T_1{}^\frown T_2$ iff there are s_1 and s_2 such that
   a) $s_1 s_2 = s$
   b) $s_1 :_{\mathbf{TYPE}_C} T_1$ if T_1 is a string type, otherwise $s_1 :_{\mathbf{TYPE}_C} {T_1}^{=1}$
   c) $s_2 :_{\mathbf{TYPE}_C} T_2$ if T_2 is a string type, otherwise $s_2 :_{\mathbf{TYPE}_C} {T_2}^{=1}$

The labelled set represented by $T_1{}^\frown T_2$ can be characterized as follows:

> Let T_1' be ${T_1}^{=1}$ if T_1 is not a string type and T_1 otherwise and similarly let T_2' be ${T_2}^{=1}$ if T_2 is not a string type and T_2 otherwise. Then $T_1{}^\frown T_2$ represents the labelled set $T_1' \cup T_2''$ where T_2'' is the result of replacing any label 'str$_i$' in $\mathrm{labels}(T_2')$ with 'str$_{i+n+1}$', where 'str$_n$' is in $\mathrm{labels}(T_1')$ and there is no j such that 'str$_j$' is in $\mathrm{labels}(T_1')$ and $j > n$.

This definition has as a consequence that '$^\frown$' is associative: $(T_1{}^\frown T_2){}^\frown T_3 = T_1{}^\frown(T_2{}^\frown T_3)$.

Concatenation of values on a defined path in a string of records

This definition is introduced in Chapter 3, p. 118.

If s is a string of length n of records such that for each i, $0 \leq i < n$, $s[i].\pi$ is a defined path, $\underset{0\leq i<n}{\mathrm{concat}}(s[i].\pi)$ denotes $s[0].\pi \ldots s[n{-}1].\pi$. We use concat$_i(s[i].\pi)$ to represent $\underset{0\leq i<\mathrm{length}(s)}{\mathrm{concat}}(s[i].\pi)$.

Predicates which relate strings

We introduce a number of distinguished predicates which are used to relate strings. The following predicates all have arity [*String*,*String*]: init, final, final_align

init "s_1 is an initial substring of s_2"

> If s_1 is a string of length n and s_2 is a string of any length, then s : init(s_1,s_2) iff the length of s_2 is greater than or equal to n and for each i, $0 \leq i < n$, $s_1[i] = s_2[i]$ and $s = s_2$.

final "s_1 is an final substring of s_2"

> If s_1 is a string of length n and s_2 is a string of length m, then s : final(s_1,s_2) iff m is greater than or equal to n and for each i, $0 \leq i < n$, $s_1[i] = s_2[(m-n)+i]$ and $s = s_2$.

final_align "s_1 is aligned with a final substring of s_2"

> If s_1:*Rec*$^+$ is a string of length n and s_2:*Rec*$^+$ is a string of length m, then s : final_align(s_1,s_2) iff

1. m is greater than or equal to n
2. s is a string of length m
3. for each i, $0 \leq i < n$,
   a) $s[(m-n)+i] : \begin{bmatrix} \mathrm{e}_1{:}\mathit{Rec} \\ \mathrm{e}_2{:}\mathit{Rec} \end{bmatrix}$
   b) $s[(m-n)+i]$.e$_1$ = $s_1[i]$
   c) $s[(m-n)+i]$.e$_2$ = $s_2[(m-n)+i]$
4. otherwise for each i, $0 \leq i < m$, $s[i] = s_2[i]$

Bibliography

Alshawi, Hiyan, ed. (1992) *The Core Language Engine*, MIT Press.

Artstein, Ron, Mark Core, David DeVault, Kallirroi Georgila, Elsi Kaiser and Amanda Stent, eds. (2011) SemDial 2011 (Los Angelogue): Proceedings of the 15th Workshop on the Semantics and Pragmatics of Dialogue.

Asudeh, Ash and Gianluca Giorgolo (2016) Perspectives, *Semantics and Pragmatics*, Vol. 9, No. 21.

Austin, J. (1962) *How to Do Things with Words*, Oxford University Press, ed. by J. O. Urmson.

Austin, J. L. (1961) Truth, in J. O. Urmson and G. J. Warnock (eds.), *J. L. Austin: Philosophical Papers*, Oxford University Press, Oxford.

Barsalou, Lawrence W. (1992a) *Cognitive psychology. An overview for cognitive scientists*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Barsalou, Lawrence W. (1992b) Frames, concepts, and conceptual fields, in A. Lehrer and E. F. Kittay (eds.), *Frames, fields, and contrasts: New essays in semantic and lexical organization*, pp. 21–74, Lawrence Erlbaum Associates, Hillsdale, NJ.

Barsalou, Lawrence W. (1999) Perceptual symbol systems, *Behavioral and Brain Sciences*, Vol. 22, pp. 577–660.

Barwise, Jon (1989) *The Situation in Logic*, CSLI Publications, Stanford.

Barwise, Jon and Robin Cooper (1981) Generalized quantifiers and natural language, *Linguistics and Philosophy*, Vol. 4, No. 2, pp. 159–219.

Barwise, Jon and Robin Cooper (1993) Extended Kamp Notation: a Graphical Notation for Situation Theory, in P. Aczel, D. Israel, Y. Katagiri and S. Peters (eds.), *Situation Theory and its Applications*, Vol. 3, CSLI, Stanford.

Barwise, Jon and John Perry (1983) *Situations and Attitudes*, Bradford Books, MIT Press, Cambridge, Mass.

Bäuerle, Rainer, Urs Egli and Arnim von Stechow, eds. (1979) *Semantics from Different Points of View* (*Springer Series in Language and Communication* 6), Springer.

Beaver, David I. and Bart Geurts (2014) Presupposition, in E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, winter 2014 edition, Metaphysics Research Lab, Stanford University. `https://plato.stanford.edu/archives/win2014/entries/presupposition/`.

Bennett, Michael Ruisdael (1974) *Some extensions of a Montague fragment of English*, PhD dissertation, UCLA. Distributed by Indiana University Linguistics Club.

van Benthem, Johan (1984) Questions about quantifiers, *Journal of Symbolic Logic*, Vol. 49, No. 2, pp. 443–466.

Blackburn, Patrick, Maarten de Rijke and Ydes Venema (2001) *Modal logic* (*Cambridge Tracts in Theoretical Computer Science* 53), Cambridge University Press.

Boas, Hans C. and Ivan A. Sag, eds. (2012) *Sign-Based Construction Grammar*, CSLI Publications.

Bos, J. (1996) Predicate logic unplugged, in P. Dekker and M. Stokhof (eds.), *Proceedings of the Tenth Amsterdam Colloquium*, pp. 133–143, ILLC/Department of Philosophy, University of Amsterdam, Amsterdam.

Botvinick, Matthew M. (2008) Hierarchical models of behavior and prefrontal function, *Trends in Cognitive Sciences*, Vol. 12, No. 5, pp. 201 – 208.

Botvinick, Matthew M., Yael Niv and Andrew C. Barto (2009) Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective, *Cognition*, Vol. 113, No. 3, pp. 262 – 280. Reinforcement learning and higher cognition.

Breitholtz, Ellen (2010) Clarification Requests as Enthymeme Elicitors, in *Aspects of Semantics and Pragmatics of Dialogue. SemDial 2010, 14th Workshop on the Semantics and Pragmatics of Dialogue* ,.

Breitholtz, Ellen (2014a) *Enthymemes in Dialogue: A micro-rhetorical approach*, PhD dissertation, University of Gothenburg.

Breitholtz, Ellen (2014b) Reasoning with topoi – towards a rhetorical approach to non-monotonicity, in *Proceedings of AISB Symposium on "Questions, discourse and dialogue: 20 years after Making it Explicit"*.

Breitholtz, Ellen (fthc) *Enthymemes and Topoi in Dialogue: An account of common sense reasoning in conversation*, Brill.

Breitholtz, Ellen and Robin Cooper (2011) Enthymemes as Rhetorical Resources, in Artstein *et al.* (2011).

Breitholtz, Ellen and Jessica Villing (2008) Can Aristotelian Enthymemes Decrease the Cognitive Load of a Dialogue System User?, in *Proceedings of LonDial 2008, the 12th SEMDIAL workshop*.

Carlson, Gregory N. (1982) Generic Terms and Generic Sentences, *Journal of Philosophical Logic*, Vol. 11, pp. 145–81.

Carnap, Rudolf (1956) *Meaning and Necessity: A Study in Semantics and Modal Logic*, second edition, University of Chicago Press.

Chatzikyriakidis, Stergios and Christine Howes, eds. (2020) Proceedings of the Probability and Meaning Conference (PaM 2020). Gothenburg: Association for Computational Linguistics. `https://www.aclweb.org/anthology/2020.pam-1.6`.

Chierchia, Gennaro (1995) *Dynamics of Meaning: Anaphora, Presupposition, and the Theory of Grammar*, University of Chicago Press, Chicago.

Chierchia, Gennaro, Barbara H. Partee and Raymond Turner, eds. (1989) *Properties, Types and Meanings*, Vol. II: Semantic Issues, Kluwer, Dordrecht.

Chierchia, Gennaro and Raymond Turner (1988) Semantics and property theory, *Linguistics and Philosophy*, Vol. 11, No. 3, pp. 261–302.

Chomsky, Noam (1981) *Lectures on Government and Binding*, Foris Publications, Dordrecht.

Christiansen, Morten and Nick Chater (2008) Language as shaped by the brain, *Behavioral and Brain Sciences*, Vol. 31, pp. 489–558.

Condoravdi, Cleo and Sven Lauer (2016) Anankastic conditionals are just conditionals, *Semantics and Pragmatics*, Vol. 9, No. 8, pp. 1–69.

Cooper, Robin (1979) The Interpretation of Pronouns, in F. Heny and H. Schnelle (eds.), *Selections from the Third Groningen Round Table* (*Syntax and Semantics* 10), pp. 61–92, Academic Press.

Cooper, Robin (1982) Binding in wholewheat* syntax (*unenriched with inaudibilia), in P. Jacobson and G. K. Pullum (eds.), *The Nature of Syntactic Representation* (*Synthese Language Library* 15), Reidel Publishing Company.

Cooper, Robin (1983) *Quantification and Syntactic Theory*, Reidel Publishing Company.

Cooper, Robin (1991) Three lectures on situation theoretic grammar, in M. Filgueiras, L. Damas, N. Moreira and A. P. Tomás (eds.), *Natural Language Processing, EAIA 90, Proceedings*, Lecture Notes in Artificial Intelligence *476*, pp. 101–140, Springer Verlag, Berlin.

Cooper, Robin (1996) The Role of Situations in Generalized Quantifiers, in S. Lappin (ed.), *The Handbook of Contemporary Semantic Theory*, Blackwell, Oxford.

Cooper, Robin (2005) Records and Record Types in Semantic Theory, *Journal of Logic and Computation*, Vol. 15, No. 2, pp. 99–112.

Cooper, Robin (2010) Frames in formal semantics, in H. Loftsson, E. Rögnvaldsson and S. Helgadóttir (eds.), *IceTAL 2010*, Springer Verlag.

Cooper, Robin (2011) Copredication, Quantification and Frames, in S. Pogodalla and J.-P. Prost (eds.), *Logical Aspects of Computational Linguistics: 6th International Conference, LACL 2011*, pp. 64–79, Springer.

Cooper, Robin (2012a) Intensional quantifiers, in T. Graf, D. Paperno, A. Szabolcsi and J. Tellings (eds.), *Theories of Everything: In Honor of Ed Keenan*, UCLA Working Papers in Linguistics *17*, pp. 69–71, Department of Linguistics, UCLA.

Cooper, Robin (2012b) Type Theory and Semantics in Flux, in R. Kempson, N. Asher and T. Fernando (eds.), *Handbook of the Philosophy of Science*, Vol. 14: Philosophy of Linguistics, pp. 271–323, Elsevier BV. General editors: Dov M. Gabbay, Paul Thagard and John Woods.

Cooper, Robin (2013a) Clarification and Generalized Quantifiers, *Dialogue and Discourse*, Vol. 4, No. 1, pp. 1–25.

Cooper, Robin (2013b) Update conditions and intensionality in a type-theoretic approach to dialogue semantics, in R. Fernández and A. Isard (eds.), *Proceedings of the 17th Workshop on the Semantics and Pragmatics of Dialogue*, pp. 15–24, University of Amsterdam.

Cooper, Robin (2014) Phrase structure rules as dialogue update rules, in V. Rieser and P. Muller (eds.), *Proceedings of The 18th Workshop on the Semantics and Pragmatics of Dialogue*, pp. 26–34.

Cooper, Robin (2015) Natural reasoning: truth or judgement based? presented at CoCoNat'15, Bloomington, Ind., 19th-20th July, 2015 `https://sites.google.com/site/typetheorywithrecords/publications/natlogtruthjudge.pdf`.

Cooper, Robin (2017a) Judgement, taste and closely related Germanic languages, in V. Rosén and K. De Smedt (eds.), *The very model of a modern linguist – in honor of Helge Dyvik*, Bergen Language and Linguistic Studies *8*, pp. 19–32, University of Bergen Library. http://dx.doi.org/10.15845/bells.v8i1.

Cooper, Robin (2017b) Proper Names in Interaction, in S. Kurahashi, Y. Ohta, S. Arai, K. Satoh and D. Bekki (eds.), *New Frontiers in Artificial Intelligence: JSAI-isAI 2016 Workshops, LENLS HAT-MASH, AI-Biz, JURISIN and SKL Kanagawa, Japan, November 14–16, 2016, Revised Selected Papers*, Lecture Notes in Artificial Intelligence *10247*, pp. 5–18, Springer.

Cooper, Robin, Simon Dobnik, Shalom Lappin and Staffan Larsson (2014a) A Probabilistic Rich Type Theory for Semantic Interpretation, in Cooper *et al.* (2014b), pp. 72–79, Association for Computational Linguistics.

Cooper, Robin, Simon Dobnik, Shalom Lappin and Staffan Larsson, eds. (2014b) *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*, Association for Computational Linguistics, Gothenburg, Sweden.

Cooper, Robin, Simon Dobnik, Shalom Lappin and Staffan Larsson (2015) Probabilistic Type Theory and Natural Language Semantics, *Linguistic Issues in Language Technology*, Vol. 10, No. 4, pp. 1–45.

Cooper, Robin and Jonathan Ginzburg (2011a) Negation in Dialogue, in Artstein *et al.* (2011), pp. 130–139.

Cooper, Robin and Jonathan Ginzburg (2011b) Negative inquisitiveness and alternatives-based negation, in *Proceedings of the Amsterdam Colloquium, 2011*.

Cooper, Robin and Jonathan Ginzburg (2012) Negative inquisitiveness and alternatives-based negation, in M. Aloni, V. Kimmelman, F. Roelofsen, G. W. Sassoon, K. Schulz and M. Westera (eds.), *Logic, Language and Meaning: 18th Amsterdam Colloquium, Amsterdam, The Netherlands, December 19–21, 2011, Revised Selected Papers*, Lecture Notes in Computer Science *7218*, pp. 32–41, Springer.

Cooper, Robin and Ruth Kempson, eds. (2008) *Language in Flux: Dialogue Coordination, Language Variation, Change and Evolution* (*Communication, Mind and Language* 1), College Publications, London.

Cooper, Robin and Aarne Ranta (2008) Natural Languages as Collections of Resources, in Cooper and Kempson (2008), pp. 109–120.

Coquand, Thierry, Randy Pollack and Makoto Takeyama (2004) A Logical Framework with Dependently Typed Records, *Fundamenta Informaticae*, Vol. XX, pp. 1–22.

Cresswell, M.J. (1985) *Structured Meanings: The Semantics of Propositional Attitudes*, MIT Press.

Dalrymple, Mary, Ronald M. Kaplan, John T. Maxwell and Annie Zaenen, eds. (1995) *Formal Issues in Lexical Functional Grammar*, CSLI Lecture Notes *47*, CSLI Publications.

Davidson, Donald (1967) The Logical Form of Action Sentences, in N. Rescher (ed.), *The Logic of Decision and Action*, University of Pittsburgh Press. Reprinted in Davidson (1980).

Davidson, Donald (1980) *Essays on Actions and Events*, Oxford University Press, New edition 2001.

Davidson, Donald and Gilbert Harman, eds. (1972) *Semantics of Natural Language*, Reidel Publishing Company.

de Bruijn, Nicolaas Govert (1972) Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem, *Indagationes Mathematicae (Proceedings)*, Vol. 75, No. 5, pp. 381–392.

Dowty, David (1989) On the semantic content of the notion of 'Thematic Role', in Chierchia *et al.* (1989), pp. 69–130.

Dowty, David, Robert Wall and Stanley Peters (1981) *Introduction to Montague Semantics*, Reidel (Springer).

Égré, Paul (2020) Logical Omniscience, in D. Gutzmann, L. Matthewson, C. Meier, H. Rullmann and T. E. Zimmermann (eds.), *The Wiley Blackwell Companion to Semantics*, John Wiley & Sons, Inc.

van Eijck, Jan and Christina Unger (2010) *Computational Semantics with Functional Programming*, Cambridge University Press.

Elbourne, Paul (2012) *Definite Descriptions*, Clarendon Press, Oxford.

Emerson, Guy (2020) Linguists Who Use Probabilistic Models Love Them: Quantification in Functional Distributional Semantics, in Chatzikyriakidis and Howes (2020), pp. 41–52, Association for Computational Linguistics. `https://www.aclweb.org/anthology/2020.pam-1.6`.

Evans, Gareth (1980) Pronouns, *Linguistic Inquiry*, Vol. 11, pp. 337–362.

Fernando, Tim (2001) Conservative Generalized Quantifiers and Presupposition, in R. Hastings, B. Jackson and Z. Zvolenszky (eds.), *Proceedings of the 11th Semantics and Linguistic Theory Conference (held May 11-13, 2001, at New York University)*, pp. 172–191.

Fernando, Tim (2004) A finite-state approach to events in natural language semantics, *Journal of Logic and Computation*, Vol. 14, No. 1, pp. 79–92.

Fernando, Tim (2006) Situations as Strings, *Electronic Notes in Theoretical Computer Science*, Vol. 165, pp. 23–36.

Fernando, Tim (2008) Finite-state descriptions for temporal semantics, in H. Bunt and R. Muskens (eds.), *Computing Meaning, Volume 3* (*Studies in Linguistics and Philosophy* 83), pp. 347–368, Springer.

Fernando, Tim (2009) Situations in LTL as strings, *Information and Computation*, Vol. 207, No. 10, pp. 980–999.

Fernando, Tim (2011) Constructing Situations and Time, *Journal of Philosophical Logic*, Vol. 40, pp. 371–396.

Fernando, Tim (2015) The Semantics of Tense and Aspect: A Finite-State Perspective, in Lappin and Fox (2015).

Fillmore, Charles J. (1970) Subjects, Speakers, and Roles, *Synthese*, Vol. 21, pp. 251–274.

Fillmore, Charles J. (1982) Frame semantics, in *Linguistics in the Morning Calm*, pp. 111–137, Hanshin Publishing Co., Seoul.

Fillmore, Charles J. (1985) Frames and the semantics of understanding, *Quaderni di Semantica*, Vol. 6, No. 2, pp. 222–254.

von Fintel, Kai and Irene Heim (2011) Intensional Semantics. MIT, Spring 2011 Edition, `http://web.mit.edu/fintel/fintel-heim-intensional.pdf`.

Fodor, Janet Dean (1970) *The Linguistic Description of Opaque Contexts*, PhD dissertation, MIT.

Fox, Chris and Shalom Lappin (2005) *Foundations of Intensional Semantics*, Blackwell Publishing.

Frege, Gottlob (1892) Über Sinn und Bedeutung, *Zeitschrift für Philosophie und philosophische Kritik*, Vol. 100, pp. 25–50. Translated in Geach and Black (1980).

Gawron, Jean Mark and Stanley Peters (1990) *Anaphora and Quantfication in Situation Semantics*, CSLI Publications.

Gazdar, Gerald, Ewan Klein, Geoffrey K Pullum and Ivan A Sag (1985) *Generalized phrase structure grammar*, Harvard University Press.

Geach, P. and M. Black, eds. (1980) *Translations from the Philosophical Writings of Gottlob Frege*, third edition, Blackwell, Oxford.

Geach, Peter Thomas (1962) *Reference and Generality: An Examination of Some Medieval and Modern Theories*, Cornell University Press.

Gibson, James J. (1979) *The Ecological Approach to Visual Perception*, Houghton, Mifflin and Company, Also available in a Classic Edition published by Psychology Press, 2015.

Gil, David (2000) Syntactic categories, cross-linguistic variation and universal grammar, in P. M. Vogel and B. Comrie (eds.), *Approaches to the typology of word classes* (*Empirical approaches to language typology* 23), Mouton de Gruyter, Berlin.

Ginzburg, Jonathan (1994) An update semantics for dialogue, in H. Bunt (ed.), *Proceedings of the 1st International Workshop on Computational Semantics*, Tilburg University.

Ginzburg, Jonathan (2010) Relevance for Dialogue, in Łupkowski and Purver (2010), pp. 121–129, Polish Society for Cognitive Science.

Ginzburg, Jonathan (2012) *The Interactive Stance: Meaning for Conversation*, Oxford University Press, Oxford.

Ginzburg, Jonathan and Robin Cooper (2004) Clarification, ellipsis, and the nature of contextual updates in dialogue, *Linguistics and Philosophy*, Vol. 27, No. 3, pp. 297–365.

Ginzburg, Jonathan and Robin Cooper (2014) Quotation via Dialogical Interaction, *Journal of Logic, Language and Information*, Vol. 23, No. 3, pp. 287–311.

Ginzburg, Jonathan, Robin Cooper and Tim Fernando (2014) Propositions, Questions, and Adjectives: a rich type theoretic approach, in Cooper *et al.* (2014b), pp. 89–96, Association for Computational Linguistics.

Ginzburg, Jonathan and Raquel Fernández (2010) Computational Models of Dialogue, in A. Clark, C. Fox and S. Lappin (eds.), *The Handbook of Computational Linguistics and Natural Language Processing*, Wiley-Blackwell.

Ginzburg, Jonathan and Massimo Poesio (2016) Grammar Is a System That Characterizes Talk in Interaction, *Frontiers in Psychology*, Vol. 7, p. 1938.

Ginzburg, Jonathan and Matt Purver (2008) Quantfication, the reprise content hypothesis, and type theory, in L. Borin and S. Larsson (eds.), *From Quantification to Conversation*, University of Gothenburg, Gothenburg.

Ginzburg, Jonathan and Ivan A. Sag (2000) *Interrogative Investigations: The Form, Meaning, and Use of English Interrogatives*, CSLI Lecture Notes *123*, CSLI Publications, Stanford, California.

Globus, Gordon G. (1995) *The Postmodern Brain* (*Advances in Concsiousness Research* 1), John Benjamins Publishing Company.

Groenendijk, Jeroen and Floris Roelofsen (2012) Course Notes on Inquisitive Semantics, NASSLLI 2012. Available at `https://sites.google.com/site/inquisitivesemantics/documents/NASSLLI-2012-inquisitive-semantics-lecture-notes.pdf`.

de Groote, Philippe and Ekaterina Lebedeva (2010) Presupposition Accommodation as Exception Handling, in *Proceedings of SIGDIAL 2010: the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 71–74.

Grosz, Barbara J., Aravind K. Joshi and Scott Weinstein (1983) Providing a unified account of definite noun phrases in discourse, in *Proceedings of ACL-83*, pp. 44–50, Cambridge, MA.

Grosz, Barbara J., Aravind K. Joshi and Scott Weinstein (1995) Centering: A framework for modeling the local coherence of discourse, *Computational Linguistics*, Vol. 21, No. 2, pp. 202–225.

Gupta, Anil (1980) *The Logic of Common Nouns: An Investigation in Quantified Model Logic*, Yale University Press, New Haven.

Halliday, M. A. K. (1977) Text as semantic choice in social contexts, in T. van Dijk and J. Petöfi (eds.), *Grammars and descriptions*, pp. 176–225, Walter de Gruyter, Berlin.

Halpern, J. (2003) *Reasoning About Uncertainty*, MIT Press, Cambridge MA.

Heim, Irene (1990) E-type Pronouns and Donkey Anaphora, *Linguistics and Philosophy*, Vol. 13, pp. 137–177.

Heim, Irene and Angelika Kratzer (1998) *Semantics in Generative Grammar*, Blackwell Publishing.

Hellan, Lars (1986) On Anaphora and Predication in Norwegian, in L. Hellan and K. K. Christensen (eds.), *Topics in Scandinavian Syntax*, Studies in Natural Language and Linguistic Theory, pp. 103–124, D. Reidel Publishing Company.

Hughes, G.E. and M.J. Cresswell (1968) *Introduction to modal logic*, Methuen and Co., Ltd.

Hughes, G.E. and M.J Cresswell (1996) *A New Introduction to Modal Logic*, Routledge.

Jackendoff, Ray (1979) How to Keep Ninety from Rising, *Linguistic Inquiry*, Vol. 10, No. 1, pp. 172–177.

Jackendoff, Ray (2002) *Foundations of Language: Brain, Meaning, Grammar, Evolution*, Oxford University Press.

Joshi, Aravind K. and Scott Weinstein (1981) Control of inference: Role of some aspects of discourse structure-centering, in *Proceedings of the IJCAI*, pp. 385–387, Vancouver, CA.

Jurafsky, Daniel and James H. Martin (2009) *Speech and Language Processing*, second edition, Pearson Education.

Kallmeyer, Laura and Rainer Osswald (2013) Syntax-driven semantic frame composition in Lexicalized Tree Adjoining Grammars, *Journal of Language Modelling*, Vol. 1, No. 2, pp. 267–330.

Kamp, Hans (1979) Events, Instants and Temporal Reference, in Bäuerle *et al.* (1979), pp. 376–418.

Kamp, Hans (1981) A Theory of Truth and Discourse Representation, in J. Groenendijk, T. Janssen and M. Stokhof (eds.), *Formal Methods in the Study of Language*, Mathematical Centre Tracts *135*, Mathematisch Centrum, Amsterdam.

Kamp, Hans (1990) Prolegomena to a Structural Theory of Belief and other Attitudes, in C. A. Anderson and J. Owens (eds.), *Propositional Attitudes: the Role of Content in Logic, Language and Mind*, CSLI Publications, Stanford.

Kamp, Hans, Josef van Genabith and Uwe Reyle (2011) Discourse Representation Theory, in D. Gabbay and F. Guenthner (eds.), *Handbook of Philosophical Logic*, Vol. 15, Springer Science+Business Media B.V. .

Kamp, Hans and Uwe Reyle (1993) *From Discourse to Logic*, Kluwer, Dordrecht.

Kanazawa, Makoto (1994) Weak vs. Strong Readings of Donkey Sentences and Monotonicity Inference in a Dynamic Setting, *Linguistics and Philosophy*, Vol. 17, pp. 109–158.

Kant, Immanuel (1781) *Critik der reinen Vernunft (Critique of Pure Reason)*, Johann Friedrich Hartknoch, Riga, second edition 1787.

Kaplan, David (1978) On the Logic of Demonstratives, *Journal of Philosophical Logic*, Vol. 8, pp. 81–98.

Keenan, E. L. and J. Stavi (1986) Natural Language Determiners, *Linguistics and Philosophy*, Vol. 9, pp. 253–326.

Kempson, Ruth, Ronnie Cann, Eleni Gregoromichelaki and Stergios Chatzikyriakidis (2016) Language as Mechanisms for Interaction, *Theoretical Linguistics*, Vol. 42, No. 3-4, pp. 203–276.

Kibble, Rodger (1997) Complement Anaphora and Dynamic Binding, in A. Lawson (ed.), *Proceedings of SALT VII*, pp. 258–275, Cornell University, Ithaca, N.Y.

King, Jeffrey C. (2014) Structured Propositions, in E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, spring 2014 edition, `http://plato.stanford.edu/archives/spr2014/entries/propositions-structured/`.

King, Jeffrey C. and Karen S. Lewis (2018) Anaphora, in E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, fall 2018 edition, Metaphysics Research Lab, Stanford University. `https://plato.stanford.edu/archives/fall2018/entries/anaphora/`.

King, Jeffrey C., Scott Soames and Jeff Speaks (2014) *New Thinking about Propositions*, Oxford University Press.

Kracht, Marcus and Udo Klein (2014) The Grammar of Code Switching, *Journal of Logic, Language and Information*, Vol. 23, pp. 313–329.

Kratzer, Angelika (1977) What 'Must' and 'Can' Must and Can Mean, *Linguistics and Philosophy*, Vol. 1, pp. 337–55.

Kratzer, Angelika (1981) The Notional Category of Modality, in H. J. Eikmeyer and H. Rieser (eds.), *Words, Worlds, and Contexts*, pp. 38–74, de Gruyter, Berlin and New York.

Kratzer, Angelika (2012) *Modals and Conditionals: New and Revised Perspectives*, Oxford University Press.

Kratzer, Angelika (2014) Situations in Natural Language Semantics, in E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, spring 2014 edition, `http://plato.stanford.edu/archives/spr2014/entries/situations-semantics/`.

Krifka, Manfred (1990) Four Thousand Ships Passed through the Lock: Object-induced Measure Functions on Events, *Linguistics and Philosophy*, Vol. 13, pp. 487–520.

Kripke, Saul (1972) Naming and Necessity, in Davidson and Harman (1972), pp. 253–355.

Kripke, Saul (1979) A Puzzle about Belief, in A. Margalit (ed.), *Meaning and Use*, Reidel.

Kölbel, Max (2004) Faultless Disagreement, *Proceedings of the Aristotelian Society*, Vol. 104, pp. 53–73.

Lappin, Shalom (2015) Curry Typing, Polymorphism, and Fine-Grained Intensionality, in Lappin and Fox (2015).

Lappin, Shalom and Chris Fox, eds. (2015) *The Handbook of Contemporary Semantic Theory*, second edition, Wiley-Blackwell.

Larson, Richard K. and Peter Ludlow (1993) Interpreted Logical Forms, *Synthese*, Vol. 96, pp. 305–55.

Larsson, Staffan (2002) *Issue-based Dialogue Management*, PhD dissertation, University of Gothenburg.

Larsson, Staffan (2010) Accommodating innovative meaning in dialogue, in Łupkowski and Purver (2010), pp. 83–90, Polish Society for Cognitive Science.

Larsson, Staffan (2013) Formal semantics for perceptual classification, *Journal of Logic and Computation*, Vol. 25, No. 2, pp. 335–369.

Larsson, Staffan and Robin Cooper (2009) Towards a formal view of corrective feedback, in A. Alishahi, T. Poibeau and A. Villavicencio (eds.), *Proceedings of the Workshop on Cognitive Aspects of Computational Language Acquisition*, pp. 1–9.

Larsson, Staffan and David R. Traum (2001) Information state and dialogue management in the TRINDI dialogue move engine toolkit, *Natural Language Engineering*, Vol. 6, No. 3&4, pp. 323–340.

Lasersohn, Peter (2005) The Temperature Paradox as Evidence for a Presuppositional Analysis of Definite Descriptions, *Linguistic Inquiry*, Vol. 36, No. 1, pp. 127–134.

Lewis, David (1972) General Semantics, in Davidson and Harman (1972), pp. 169–218.

Lewis, David (1973) *Counterfactuals*, Harvard University Press, Revised printing 1986.

Lewis, David (1979a) Attitudes de dicto and de se, *Philosophical Review*, Vol. 88, pp. 513–543. Reprinted in Lewis (1983).

Lewis, David (1979b) Scorekeeping in a Language Game, *Journal of Philosophical Logic*, Vol. 8, pp. 339–359.

Lewis, David (1983) *Philosophical Papers, Volume 1*, Oxford University Press.

Lewis, David K. (1981) Ordering Semantics and Premise Semantics for Counterfactuals, *Journal of Philosophical Logic*, Vol. 10, pp. 217–234.

Linell, Per (2009) *Rethinking Language, Mind, and World Dialogically: Interactional and contextual theories of human sense-making*, Advances in Cultural Psychology: Constructing Human Development, Information Age Publishing, Inc., Charlotte, N.C.

Löbner, Sebastian (1979) *Intensionale Verben und Funktionalbegriffe. Untersuchung zur Syntax und Semantik von* wechseln *und den vergleichbaren Verben des Deutschen*, Narr, Tübingen.

Löbner, Sebastian (1981) Intensional Verbs and Functional Concepts: More on the "Rising Temperature" Problem, *Linguistic Inquiry*, Vol. 12, No. 3, pp. 471–477.

Löbner, Sebastian (2014) Evidence for frames from human language, in T. Gamerschlag, D. Gerland, W. Petersen and R. Osswald (eds.), *Frames and Concept Types* (*Studies in Linguistics and Philosophy* 94), pp. 23–68, Springer, Heidelberg, New York.

Löbner, Sebastian (2015) Functional Concepts and Frames, in T. Gamerschlag, D. Gerland, R. Osswald and W. Petersen (eds.), *Meaning, Frames, and Conceptual Representation*, pp. 15–42, Düsseldorf University Press.

Löbner, Sebastian (in prep) Functional Concepts and Frames. Available from `http://semanticsarchive.net/Archive/jI1NGEwO/Loebner_Functional_Concepts_and_Frames.pdf`.

Lücking, Andy and Jonathan Ginzburg (2019) Not few but all quantifiers can be negated: Towards a referentially transparent semantics of quantified noun phrases, in *Proceedings of the Amsterdam Colloquium*, pp. 269–278.

Ludlow, Peter (2014) *Living Words: Meaning Underdetermination and the Dynamic Lexicon*, Oxford University Press.

Łupkowski, Paweł and Matthew Purver, eds. (2010) Aspects of Semantics and Pragmatics of Dialogue. SemDial 2010, 14th Workshop on the Semantics and Pragmatics of Dialogue. Poznań: Polish Society for Cognitive Science.

Maier, Emar (2009) Proper names and indexicals trigger rigid presuppositions, *Journal of Semantics*, Vol. 26, pp. 253–315.

Maier, Emar (2016) Attitudes and Mental Files in Discourse Representation Theory, *Review of Philosophy and Psychology*, Vol. 7, No. 2, pp. 473–490.

Maier, Emar (2017) Referential Dependencies Between Conflicting Attitudes, *Journal of Philosophical Logic*, Vol. 46, No. 2, pp. 141–167.

Martin-Löf, Per (1984) *Intuitionistic Type Theory*, Bibliopolis, Naples.

Mates, Benson (1952) Synonymity, in L. Linsky (ed.), *Semantics and the Philosophy of Language*, pp. 111–136, University of Illinois Press, Urbana, IL. First published 1950 in *University of California Publications in Philosophy*, 25.

McCawley, James D. (1979) Presupposition and Discourse Structure, in C.-K. Oh and D. A. Dinneen (eds.), *Presupposition* (*Syntax and Semantics* 11), Academic Press.

Menzel, Christopher (2015) Possible Worlds, in E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, summer 2015 edition, `http://plato.stanford.edu/archives/sum2015/entries/possible-worlds/`.

Montague, Richard (1970) Universal Grammar, *Theoria*, Vol. 36, pp. 373–398.

Montague, Richard (1973) The Proper Treatment of Quantification in Ordinary English, in J. Hintikka, J. Moravcsik and P. Suppes (eds.), *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*, pp. 247–270, D. Reidel Publishing Company, Dordrecht.

Montague, Richard (1974) *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven, ed. and with an introduction by Richmond H. Thomason.

Moxey, Linda and Anthony Sanford (1987) Quantifiers and focus, *Journal of Semantics*, Vol. 5, pp. 189–206.

Ninan, Dilip (2010) *De Se* Attitudes: Ascription and Communication, *Philosophy Compass*, Vol. 5, No. 7, pp. 551–567.

Nordström, Bengt, Kent Petersson and Jan M. Smith (1990) *Programming in Martin-Löf's Type Theory* (*International Series of Monographs on Computer Science* 7), Clarendon Press, Oxford.

Nouwen, Rick (2003) Complement Anaphora and Interpretation, *Journal of Semantics*, Vol. 20, No. 1, pp. 73–113.

Partee, Barbara H. (1977) Possible World Semantics and Linguistic Theory, *The Monist*, Vol. 60, No. 3, pp. 303–326.

Partee, Barbara H. (1979) Semantics – Mathematics or Psychology?, in Bäuerle *et al.* (1979).

Partee, Barbara H. (1986) Noun Phrase Interpretation and Type-Shifting Principles, in J. Groenendijk, D. de Jongh and M. Stokhof (eds.), *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers*, Foris Publications.

Partee, Barbara H. (2014) The History of Formal Semantics: Changing Notions of Linguistic Competence. 9th Annual Joshua and Verona Whatmough Lecture, Harvard, `https://udrive.oit.umass.edu/partee/Partee2014Harvard.pdf`, `https://www.youtube.com/watch?v=0VV-1NDKmEc`.

Partee, Barbara H., Alice G.B. ter Meulen and Robert E. Wall (1990) *Mathematical Methods in Linguistics*, Springer.

Pelletier, Jeff and Lenhard Schubert (1989) Generically Speaking, in Chierchia *et al.* (1989), pp. 193–268.

Perry, John (1979) The Problem of the Essential Indexical, *Noûs*, Vol. 13, No. 1, pp. 3–21. Reprinted in Perry (1993).

Perry, John (1993) *The Problem of the Essential Indexical and Other Essays*, Oxford University Press.

Peters, Stanley and Dag Westerståhl (2006) *Quantifiers in Language and Logics*, Oxford University Press.

Poesio, Massimo, Rosemary Stevenson, Barbara Di Eugenio and Janet Hitzeman (2004) Centering: A Parametric Theory and Its Instantiations, *Computational Linguistics*, Vol. 30, No. 3, pp. 309–363.

Portner, Paul (2009) *Modality* (*Oxford Surveys in Semantics and Pragmatics* 1), Oxford University Press.

Potts, Christopher (2005) *The Logic of Conventional Implicatures*, Oxford University Press.

Prinz, Jesse J. and Lawrence W. Barsalou (2014) Steering a Course for Embodied Representation, in E. Dietrich and A. B. Markman (eds.), *Cognitive Dynamics: Conceptual and Representational Change in Humans and Machines*, pp. 51–77, Psychology Press. Previously published in 2000 by Lawrence Erlbaum.

Pross, Tillmann (ms) Fodor's puzzle and the semantics of attitude reports. Available from `http://www.ims.uni-stuttgart.de/institut/mitarbeiter/prosstn/files/pross-attitudes.pdf`.

Purver, Matt and Jonathan Ginzburg (2004) Clarifying noun phrase semantics, *Journal of Semantics*, Vol. 21, No. 3, pp. 283–339.

Purver, Matthew, Eleni Gregoromichelaki, Wilfried Meyer-Viol and Ronnie Cann (2010) Splitting the *I*s and Crossing the *You*s: Context, Speech Acts and Grammar, in Łupkowski and Purver (2010), pp. 43–50, Polish Society for Cognitive Science.

Quine, W. V. (1948) On what there is, *Review of Metaphysics*, Vol. ??, No. ?? reprinted in From a Logical Point of View (Cambridge, Massachusetts; Harvard University Press: 1953).

Ranta, Aarne (1994) *Type-Theoretical Grammar*, Clarendon Press, Oxford.

Recanati, François (2010) *Truth-Conditional Pragmatics*, Clarendon Press Oxford.

Rescher, Nicholas (1999) How Many Possible Worlds Are There?, *Philosophy and Phenomenological Research*, Vol. 59, No. 2, pp. pp. 403–420.

Rey, Georges (2015) The Analytic/Synthetic Distinction, in E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, fall 2015 edition, `http://plato.stanford.edu/archives/fall2015/entries/analytic-synthetic/`.

Reyle, Uwe (1993) Dealing with ambiguities by underspecification: Construction, representation and deduction, *Journal of Semantics*, Vol. 10, No. 2, pp. 123–179.

Ribas-Fernandes, José J.F., Alec Solway, Carlos Diuk, Joseph T. McGuire, Andrew G. Barto, Yael Niv and Matthew M. Botvinick (2011) A Neural Signature of Hierarchical Reinforcement Learning, *Neuron*, Vol. 71, No. 2, pp. 370 – 379.

Romero, Maribel (2008) The Temperature Paradox and Temporal Interpretation, *Linguistic Inquiry*, Vol. 39, No. 4, pp. 655–667.

Romoli, Jacopo and Yasutada Sudo (2009) De Re / De Dicto Ambiguity and Presupposition Projection, in A. Riester and T. Solstad (eds.), *Proceedings of Sinn und Bedeutung 13*, pp. 425–438. `http://semanticsarchive.net/Archive/DhhOTI2Z/sub13proc.pdf`.

Ruppenhofer, Josef, Michael Ellsworth, Miriam R.L. Petruck, Christopher R. Johnson and Jan Scheffczyk (2006) FrameNet II: Extended Theory and Practice. Available from the FrameNet website.

Russell, Betrand (1903) *Principles of Mathematics*, Cambridge University Press.

Sacks, H., E.A. Schegloff and G. Jefferson (1974) A simplest systematics for the organization of turn-taking for conversation, *Language*, Vol. 50, pp. 696–735.

Sag, Ivan A., Thomas Wasow and Emily M. Bender (2003) *Syntactic Theory: A Formal Introduction*, 2nd edition, CSLI Publications, Stanford.

Van der Sandt, Rob A (1992) Presupposition projection as anaphora resolution, *Journal of semantics*, Vol. 9, No. 4, pp. 333–377.

Sanford, Anthony and Linda Moxey (1993) *Communicating quantities: A psychological perspective*, Laurence Erlbaum Associates.

de Saussure, Ferdinand (1916) *Cours de linguistique générale*, Payot, Lausanne and Paris, edited by Charles Bally and Albert Séchehaye.

Schlenker, Philippe (2011) Indexicality and *De Se* Reports, in C. Maienborn, K. v. Heusinger and P. Portner (eds.), *Semantics: an international handbook of natural language meaning*, pp. 1561–1604, de Gruyter.

Schubert, Lenhart K. (2000) The Situations We Talk about, in J. Minker (ed.), *Logic-Based Artificial Intelligence*, pp. 407–439, Kluwer Academic Publishers, Dortrecht.

Schwager, Magdalena (2009) Speaking of qualities, in E. Cormany, S. Ito and D. Lutz (eds.), *Proceedings of SALT 19*.

Searle, John R. (1969) *Speech Acts: an Essay in the Philosophy of Language*, Cambridge University Press.

Seligman, Jerry and Larry Moss (1997) Situation Theory, in J. van Benthem and A. ter Meulen (eds.), *Handbook of Logic and Language*, North Holland and MIT Press.

Shieber, Stuart (1986) *An Introduction to Unification-Based Approaches to Grammar*, CSLI Publications, Stanford.

Steedman, Mark (2012) *Taking Scope: The Natural Semantics of Quantifiers*, The MIT Press.

Suppes, Patrick (1960) *Axiomatic Set Theory*, The University Series in Undergraduate Mathematics, D. van Nostrand Company, Inc.

Sutton, Peter R. and Hana Filip (2017) Individuation, reliability, and the mass/count distinction, *Journal of Language Modelling*, Vol. 5, No. 2, pp. 303–356. DOI: http://dx.doi.org/10.15398/jlm.v5i2.144.

Thomason, Richmond H. (1979) Home is where the heart is, in P. A. French, Uehling, Jr., Theodore E. and H. K. Wettstein (eds.), *Contemporary perspectives in the philosophy of language*, pp. 209–219, University of Minnesota Press, Minneapolis.

Thomason, Richmond H. (1980) A model theory for propositional attitudes, *Linguistics and Philosophy*, Vol. 4, pp. 47–70.

Traum, David R. (1994) *A Computational Theory of Grounding in Natural Language Conversation*, PhD dissertation, University of Rochester, Department of Computer Science.

Turner, Raymond (2005) Semantics and Stratification, *Journal of Logic and Computation*, Vol. 15, No. 2, pp. 145–158.

Wadler, Philip (2015) Propositions as Types, *Communications of the ACM*, Vol. 58, No. 12, pp. 75–84.

Walker, Marilyn A., Aravind K. Joshi and Ellen F. Prince, eds. (1998) *Centering Theory in Discourse*, Oxford University Press, Oxford.

Zweig, Eytan (2008) *Dependent Plurals and Plural Meaning*, PhD dissertation, New York University.

Zweig, Eytan (2009) Number-neutral bare plurals and the multiplicity implicature, *Linguistics and Philosophy*, Vol. 32, pp. 353–407.