well-formed predication: *John wanted to bank his check that very day* or *Mary wanted to bank the airplane*.

4.2 The basic system of types

Let us begin by examining simple, functional, and disjunctive types, as well as the types that carry presuppositions. Thus, we have

- **Simple or Primitive Types**: E, T, PHYSICAL OBJECT, etc.
- **Presuppositional Type**: Π, another base type that carries the type presuppositions of terms.
- **Disjunctive Types:** If σ, τ, and ρ are types, $\sigma \sqsubseteq \rho$ and $\tau \sqsubseteq \rho$, then $(\sigma \vee \tau)$ is a type.
- **Functional Types:** If σ and τ are types, then so is $(\sigma \Rightarrow \tau)$.
- **Quantificational Types:** If σ is a simple type, and τ is any expression denoting a type and x is a variable ranging over types, then $\exists \mathrm{x} \sqsubseteq \sigma\ \tau$ is a type.[8] To illustrate, a term *t* is of this quantificational type if there is a subtype x of σ such that *t* is of type $\tau[\mathrm{x}]$.

The set of simple types, ST, forms the core of the system. It contains the basic types countenanced by Montague Grammar, E, the general type of entities, and T, the type of propositions, along with a finite set of subtypes of E and a countable set of subtypes of T. Another distinguished subtype is $\bot$, the absurd type. When there is no type in the type hierarchy γ such that $\alpha \sqsubseteq \gamma$ and $\beta \sqsubseteq \gamma$, $\alpha \vee \beta$ represents a disjunctive object that is the internal semantics of an accidentally polysemous term that must be resolved to assign the term a determinate type. Functional types represent properties. ST comes with the subtyping relation $\sqsubseteq$, which forms a semi-lattice over ST with $\bot$ at the base.[9]

Using $\sqsubseteq$ on the simple types, we define a greatest lower bound operation $\sqcap$ for elements of ST.

Definition 2 *Greatest Lower Bound:* $\alpha \sqcap \alpha' = \beta$ *iff* $\beta \sqsubseteq \alpha$ *and* $\beta \sqsubseteq \alpha'$ *and there is no* $\gamma \neq \beta$ *such that* $\beta \sqcap \gamma$ *and* $\gamma \sqsubseteq \alpha$ *and* $\gamma \sqsubseteq \alpha'$.

$\sqcap$ has the usual properties—e.g., idempotence, commutativity, and $\alpha \sqsubseteq \beta$ iff $\alpha \sqcap \beta = \alpha$. TCL captures incompatibility between types in terms of their

[8] x is, I realize, close to an individual level variable *x*. I strived for typographic consistency making all type formulas in small caps. Hopefully this will not cause too much confusion.

[9] As we will see, the subtyping relation as defined in the next section will entail that $\top$ is not the supremum of the lattice. In fact $\top$ is not a type. Note that the fact that $\top$ is not a type in the hierarchy does not stop us from using the tautology $\top$ in logical forms. These are quite different objects.

common join, $\perp$. We can also define a dual to greatest lower bound, least upper bound, or $\sqcup$: $\alpha \sqcup \beta = \gamma$ iff γ is the least general type in the hierarchy such that $\alpha \sqcup \gamma$ and $\beta \sqcup \gamma = \gamma$. Note that $\sqcup$ may not be always defined, since there may be no type that is the least upper bound of arbitrary types α and β

4.2.1 Subtyping

In the previous chapter, we saw that the standard set theoretic model of types fails to provide a coherent notion of subtyping for functional types, once we admit a rich set of subtypes of the type of entities E. To summarize the difficulty, recall that according to set theory, the set of physical properties or functions of type P $\Rightarrow$ T, that is, the set of all functions from objects of PHYSICAL OBJECT type to propositions, and the set of first-order properties or functions of type E $\Rightarrow$ T (the set of all functions from entities to propositions) are disjoint, even though P $\sqsubseteq$ E in the lattice of simple types and even though every function in P $\Rightarrow$ T is a subfunction of some function in E $\Rightarrow$ T. There is no coherent notion of subtyping for higher-order types, where subtype is understood as subset, once we admit multiple subtypes of E.

Type theory and the categorial models that I develop below provide a coherent notion of subtyping for all types, in the sense that, together with the rules of the simple, typed λ calculus, they generate a consistent logic or system of proof. We need such a notion of subtyping to specify an appropriate rule of application for β reduction: roughly one can apply a λ term $\lambda x \phi$ to a term t if the type of t is a subtype of the type of x. I will specify subtying using a restricted, intuitionistic notion of deduction or proof for types, $\vdash_\Delta$.

- From subtyping to logic:

$$\frac{\alpha \sqsubseteq_{\mathrm{ST}} \beta}{\alpha \vdash_\Delta \beta}$$

In particular, the model will verify:

Fact 1 *Subtyping for functional types:*

$$\frac{\alpha \sqsubseteq \alpha' \quad \beta \sqsubseteq \beta'}{(\alpha' \Rightarrow \beta) \sqsubseteq (\alpha \Rightarrow \beta')}$$

Subtyping for functional types implies that E $\Rightarrow$ T $\sqsubseteq$ P $\Rightarrow$ T. This makes sense from a proof theoretic or computational point of view: if you have a proof that given a proof of an entity, you have the proof of some proposition, then you have a proof that given a proof of an entity of a particular type (say a physical object), you have a proof of a proposition. But we cannot derive $\mathrm{P} \Rightarrow \mathrm{T} \sqsubseteq \mathrm{E} \Rightarrow \mathrm{T}$. This seems not to get us what we want for our type hierarchy, since this implies, on the usual conception of first-order properties, that the type of first-order properties is a subtype of the type of physical properties.[10]

In light of this, we must re-examine what we mean by a first-order property. In a system with many subtypes of E, something is a first-order property just in case it is a function from some subtype of E into the type of propositions. To spell this out, our types must be defined in a second-order language for types. The type of first-order property would thus not be what we naively take it to be, namely $\mathrm{E} \Rightarrow \mathrm{T}$, but rather something that is implied by all function types taking as inputs subtypes of E and returning a proposition. That is, the type of a first-order property is:

(4.4) $\exists \mathrm{x} \sqsubseteq \mathrm{E}\ (\mathrm{x} \Rightarrow \mathrm{T})$

Anything from whose type declaration we can "prove" (4.4) is a first-order property. To get anywhere, we must provide subtyping rules for existentially quantified types. To get a sensible notion of subtyping as deduction, my subtyping rules follow the standard introduction and elimination rules for $\exists$. In particular, where A is any type expression with an occurrence of β and B a type expression where β does not occur, then

- Type theoretic $\exists$ introduction:

$$\frac{\beta \sqsubseteq \alpha}{A \ \sqsubseteq\ (\exists \mathrm{x} \sqsubseteq \alpha\ A(\frac{X}{\beta}))}$$

- Type theoretic $\exists$ "exploitation":

$$\frac{\beta \sqsubseteq \alpha,\ A \sqsubseteq B}{(\exists \mathrm{x} \sqsubseteq \alpha\ A(\frac{\mathrm{X}}{\beta}))\ \sqsubseteq\ B}$$

This enables us to get the right facts about first-order properties. In particular, take the λ expression for *black dog*, whose course grained, denotational meaning is a function from physical objects to propositions. The NP has the type $\mathrm{P} \Rightarrow \mathrm{T}$, from which we can easily prove (4.4) using the $\exists$ introduction rule. We can now combine *black dog* with a determiner whose type presupposition

[10] This has disastrous consequences for the construction of logical form. Consider the rule of application in the λ calculus which is like Modus Ponens—given a type α and a type $\alpha \Rightarrow \gamma$, we get γ. Now take the case of a determiner which is something of type $(\mathrm{E} \Rightarrow \mathrm{T}) \Rightarrow ((\mathrm{E} \Rightarrow \mathrm{T}) \Rightarrow \mathrm{T})$ and it must combine with something of $\mathrm{P} \Rightarrow \mathrm{T}$. We have by assumption that $\mathrm{E} \Rightarrow \mathrm{T} \vdash \mathrm{P} \Rightarrow \mathrm{T}$. But we cannot now apply the determiner meaning to its restrictor; application is not sound in this case, just as $\beta \vdash \alpha$ does not allow us to conclude: $\beta \rightarrow \gamma, \alpha \vdash \gamma$.

on its first argument is that given by (4.4). We also have the general type of physical properties, $\exists x \sqsubseteq \textsc{p}\ (x \Rightarrow \textsc{t})$, the general type of informational properties, $\exists x \sqsubseteq \textsc{i}\ (x \Rightarrow \textsc{t})$, and so on. The subtype hierarachy for these will be the intuitive one.

(4.4) is the type presupposition of anything that intuitively takes a first-order property as an argument—e.g., a determiner or DP. Any expression that expresses a particular first-order property will satisfy this presupposition in the sense of entailing it. Thus:

Fact 2 *Any ordinary physical property (e.g., mass, shape, weight, color, etc.) is a first-order property and any property of informational objects (e.g., the property of being interesting, intelligible, etc.) is a first-order property.*

In addition, applying a physical property to an object of non-physical type is not defined (yields a type clash), and similarly applying a property defined only on entities of abstract object type, i.e., of type I, to something of type P is not defined.

4.3 Lexical entries and type presuppositions

In the simply typed lambda calculus, type checking is done automatically during the moment of application. In the system developed here, however, a clash between the type requirements of a predicate and the types of its arguments may require adjustments to the predication relation and to logical form. Doing this directly within the typed λ calculus led Asher and Pustejovsky (2006) to unwanted complexity, and so I have chosen a different route, separating out those operations involving type presupposition justification from the core of the simply typed λ calculus. To pass presuppositions through properly from predicates to arguments, I add a presuppositional parameter to each type as de Groote (2006) and Pogodalla (2008) do to handle dynamic contexts.[11]

Each term has an extra argument for a presupposition element that can be modified by the lexical item. For instance, the standard lexical semantic entry for *tree* looks like this:

(4.5) $\lambda x{:}\ \textsc{p}\ \mathrm{tree}(x)$

[11] Since I'm not trying to embed dynamic semantics in the λ calculus, I do not resort to their continuation style semantics. They add two parameters of interpretation, but I shall add only one. I use standard dynamic semantics for passing type values across discourse spans. Nevertheless, everything I do here should be fully compatible with other approaches to dynamic semantics.