

Session 5: Quantifiers and Scope

Where Composition Stops Being Mechanical

James Pustejovsky

Brandeis University
Department of Computer Science
CS 135: Computational Semantics

September 11, 2026

Where We Were

Sessions 1–4 built a working fragment. Types drive composition, function application does the work, and every sentence gets a truth value against a model.

Three loose ends were left deliberately.

- ① **Session 2:** proper names are type e , quantified NPs are $\langle\langle e, t \rangle, t\rangle$. Two types, one syntactic category.
- ② **Session 3:** $\forall\exists \neq \exists\forall$, with a counter-model. What does that mean for English?
- ③ **Session 4:** rule (1), $S \rightarrow NP VP$, has *two* composition paths. That violates rule-to-rule.

All three are the same loose end. [Today we tie it off](#) — and in the process meet the phenomenon that makes semantic composition genuinely hard.

Today

- ① Generalized quantifiers: NPs as sets of sets
- ② What determiners can mean — conservativity, monotonicity
- ③ Why every NP should have the same type: the coordination argument
- ④ Scope ambiguity: one structure, two readings
- ⑤ Three mechanisms — QR, storage, continuations
- ⑥ Entailment between readings, and which readings actually occur
- ⑦ What this means for Problem Set 1

This is the capstone of Part I. It is also the session that Sessions 2, 3, and 4 have each been deferring to.

The Problem, Restated

Alice sleeps.

Every student sleeps.

Same syntax: $S \rightarrow NP VP$. If the semantics is compositional, the *same rule* must derive both.

But *every student* cannot denote an individual:

- there is no one person it picks out;
- *no student* would have to pick out a non-existent individual;
- and *Every student sleeps* does not mean sleep(someone in particular).

The Problem, Restated

Alice sleeps.

Every student sleeps.

Same syntax: $S \rightarrow NP VP$. If the semantics is compositional, the *same rule* must derive both.

But *every student* cannot denote an individual:

- there is no one person it picks out;
- *no student* would have to pick out a non-existent individual;
- and *Every student sleeps* does not mean sleep(someone in particular).

Montague's move: if the NP cannot be the argument, make it the *function*.

This inverts the naive picture, in which predicates apply to things. Here a noun phrase applies to a predicate. It takes some getting used to, and it is correct.

A Quantifier Is a Set of Sets

$$\llbracket \text{every student} \rrbracket = \lambda P. \forall x [\text{student}(x) \rightarrow P(x)] \quad : \langle \langle e, t \rangle, t \rangle$$

Equivalently, and more revealingly: **the set of all sets that contain every student.**

NP	Denotes the set of sets X such that...
<i>every student</i>	$\text{student} \subseteq X$
<i>some student</i>	$\text{student} \cap X \neq \emptyset$
<i>no student</i>	$\text{student} \cap X = \emptyset$
<i>most students</i>	$ \text{student} \cap X > \text{student} - X $
<i>Alice</i>	$\text{Alice} \in X$

Note the last row. A proper name has a perfectly good reading as a set of sets: *the properties Alice has*. That is Partee's LIFT from Session 2, and it is about to become important.

Determiners

If *every student* is $\langle\langle e, t \rangle, t\rangle$ and *student* is $\langle e, t \rangle$, the determiner must be:

$$\llbracket \text{every} \rrbracket = \lambda Q. \lambda P. \forall x [Q(x) \rightarrow P(x)] \quad : \langle\langle e, t \rangle, \langle\langle e, t \rangle, t\rangle\rangle$$

A relation between two sets — the **restrictor** Q and the **scope** P .

Determiner	Denotation	Relation between Q and P
<i>every</i>	$\lambda Q \lambda P. \forall x [Q(x) \rightarrow P(x)]$	$Q \subseteq P$
<i>some, a</i>	$\lambda Q \lambda P. \exists x [Q(x) \wedge P(x)]$	$Q \cap P \neq \emptyset$
<i>no</i>	$\lambda Q \lambda P. \neg \exists x [Q(x) \wedge P(x)]$	$Q \cap P = \emptyset$
<i>at least two</i>	—	$ Q \cap P \geq 2$

Viewing determiners as *relations between sets* is what makes the next section possible: we can now ask which relations are possible determiner meanings, and the answer is surprisingly restrictive.

How Many Determiners Are There?

A determiner is a relation between two subsets of $\mathcal{D}$. With $|\mathcal{D}| = n$:

$$\text{possible determiner meanings} = 2^{(2^n \times 2^n)} = 2^{4^n}$$

For $n = 2$ that is already $2^{16} = 65,536$.

But natural languages use almost none of them. No language has a determiner meaning, say,

“Q and P have the same number of elements, or Alice is in P”

So: which relations *are* possible determiner meanings? This is one of the genuine empirical successes of formal semantics — the constraints are sharp, cross-linguistically robust, and have real consequences.

Barwise & Cooper (1981); Keenan & Stavi (1986); van Benthem (1986).

CONS

$$D(Q)(P) \iff D(Q)(Q \cap P)$$

Only the part of the scope inside the restrictor matters.

Every student sleeps $\equiv$ *Every student is a student who sleeps*

Some student sleeps $\equiv$ *Some student is a student who sleeps*

No student sleeps $\equiv$ *No student is a student who sleeps*

These paraphrases are not merely true — they are **trivially, boringly** true. That triviality is the phenomenon.

Every determiner in every language studied is conservative. It is the strongest candidate for a semantic universal we have.

The Test Case: *only*

Only students sleep vs. *Only students are students who sleep*

These are not equivalent.

- The first says: everyone who sleeps is a student. It constrains *non*-students.
- The second is trivially true — everything that is a student-who-sleeps is a student.

So *only* is not conservative.

The standard conclusion: **only is not a determiner**. It is a focus-sensitive adverbial that happens to appear in determiner position. Note the shape of the argument — a semantic universal is used to make a syntactic classification. That is the kind of leverage a good universal gives you.

Monotonicity

Determiners also differ in how they behave under *enlarging* or *shrinking* their arguments.

Upward and downward monotone (in the scope argument)

$\text{MON}\uparrow: D(Q)(P) \text{ and } P \subseteq P' \implies D(Q)(P')$

$\text{MON}\downarrow: D(Q)(P) \text{ and } P' \subseteq P \implies D(Q)(P')$

Determiner	Scope	Example inference
every	$\text{MON}\uparrow$	<i>every student jogs</i> $\Rightarrow$ <i>every student moves</i>
some	$\text{MON}\uparrow$	<i>some student jogs</i> $\Rightarrow$ <i>some student moves</i>
no	$\text{MON}\downarrow$	<i>no student moves</i> $\Rightarrow$ <i>no student jogs</i>
exactly two	neither	no inference either way

Monotonicity is the engine of natural-language inference, and it is computable directly from the determiner's denotation.

Why Monotonicity Matters

Negative polarity items are licensed exactly in downward-monotone contexts.

	Sentence	Determiner
ok	No student had <u>any</u> money	MON $\downarrow$
*	Every student had <u>any</u> money	MON $\uparrow$ (in scope)
ok	<u>Every</u> student who had <u>any</u> money left	MON $\downarrow$ in restrictor

The third row is the striking one. *Every* is upward monotone in its scope but **downward** monotone in its restrictor — and *any* is licensed exactly there.

A semantic property predicts a **syntactic distribution**, argument position by argument position. This is as good as evidence gets that the generalized-quantifier analysis is tracking something real, and not just a convenient encoding.

Worked Derivation: *Every student sleeps*

Step 1 — determiner meets restrictor.

$$\begin{aligned}\llbracket \text{every student} \rrbracket &= (\lambda Q. \lambda P. \forall x [Q(x) \rightarrow P(x)]) (\lambda z. \text{student}(z)) \\ &\rightsquigarrow \lambda P. \forall x [(\lambda z. \text{student}(z))(x) \rightarrow P(x)] \\ &\rightsquigarrow \lambda P. \forall x [\text{student}(x) \rightarrow P(x)] \quad : \langle \langle e, t \rangle, t \rangle\end{aligned}$$

Step 2 — quantifier meets scope.

$$\begin{aligned}\llbracket \text{Every student sleeps} \rrbracket &= (\lambda P. \forall x [\text{student}(x) \rightarrow P(x)]) (\lambda w. \text{sleep}(w)) \\ &\rightsquigarrow \forall x [\text{student}(x) \rightarrow (\lambda w. \text{sleep}(w))(x)] \\ &\rightsquigarrow \forall x [\text{student}(x) \rightarrow \text{sleep}(x)] \quad : t\end{aligned}$$

Four β -reductions, and the composition rule was function application throughout.

Note step 2: the *subject* is the function and the *VP* is the argument. The types said so; we did not have to stipulate it.

Practice: Write the Quantifier

Give the denotation, of type $\langle\langle e, t \rangle, t \rangle$, for each NP.

- ① *no dog*
- ② *some student*
- ③ *Alice* (lifted)
- ④ *every sleeping student*
- ⑤ *at least two books*

Practice: Write the Quantifier

Give the denotation, of type $\langle\langle e, t \rangle, t\rangle$, for each NP.

- ① *no dog*
- ② *some student*
- ③ *Alice* (lifted)
- ④ *every sleeping student*
- ⑤ *at least two books*

Answers.

- ① $\lambda P. \neg \exists x [\text{dog}(x) \wedge P(x)]$
- ② $\lambda P. \exists x [\text{student}(x) \wedge P(x)]$
- ③ $\lambda P. P(a)$
- ④ $\lambda P. \forall x [(\text{student}(x) \wedge \text{sleep}(x)) \rightarrow P(x)]$
- ⑤ $\lambda P. |\{x : \text{book}(x) \wedge P(x)\}| \geq 2$

Item 4 shows predicate modification feeding the restrictor. Item 5 is not first-order expressible with a single quantifier — counting genuinely needs the cardinality operator.

Two More Determiner Properties

Symmetry

$$D(Q)(P) \iff D(P)(Q)$$

Determiner	Sym.?	Test
<i>some</i>	yes	<i>some students are athletes</i> $\equiv$ <i>some athletes are students</i>
<i>no</i>	yes	<i>no students are athletes</i> $\equiv$ <i>no athletes are students</i>
<i>every</i>	no	<i>every student is an athlete</i> $\not\equiv$ <i>every athlete is a student</i>

Symmetric determiners are exactly those licensed in *there*-sentences: *there are some/no/two students*, but **there is every student*.

Existential import

Every unicorn sleeps comes out **true** — vacuously, since there are no unicorns. Whether every presupposes a non-empty restrictor is contested, and goes back to Aristotle.

The Type Mismatch, Revisited

Currently our fragment has two composition paths for one syntactic rule:

Subject	Type	Direction of application
<i>Alice</i>	e	VP applies to NP
<i>every student</i>	$\langle\langle e, t \rangle, t\rangle$	NP applies to VP

Session 4 called this the fragment's one real design tension, and deferred the argument. Here it is.

Two ways out:

- ① Keep both paths, and let the rule be disjunctive. (What PS 1 does.)
- ② **Lift every NP** to $\langle\langle e, t \rangle, t\rangle$, so one path suffices.

Option 2 costs something — proper names get a more complicated denotation than they need. So we need a real argument for paying that cost.

The Coordination Argument

Alice and every student left.

Coordination requires its two conjuncts to have the **same type**. You cannot conjoin a truth value with an individual.

<i>Alice</i>	<i>e</i>
<i>every student</i>	$\langle \langle e, t \rangle, t \rangle$

These cannot be conjoined as they stand. And yet the sentence is perfectly good English.

The Coordination Argument

Alice and every student left.

Coordination requires its two conjuncts to have the **same type**. You cannot conjoin a truth value with an individual.

<i>Alice</i>	<i>e</i>
<i>every student</i>	$\langle \langle e, t \rangle, t \rangle$

These cannot be conjoined as they stand. And yet the sentence is perfectly good English.

Lift the name using $\text{LIFT}(x) = \lambda P. P(x)$:

$$\llbracket \text{Alice} \rrbracket = \lambda P. P(a) \quad : \langle \langle e, t \rangle, t \rangle$$

Now both conjuncts are $\langle \langle e, t \rangle, t \rangle$, and conjunction proceeds:

$$\llbracket \text{Alice and every student} \rrbracket = \lambda P. P(a) \wedge \forall x[\text{student}(x) \rightarrow P(x)]$$

What Lifting Buys

	Two types	Uniform lifting
$S \rightarrow NP VP$	two paths	one path
Coordination of NPs	fails	works
Rule-to-rule hypothesis	violated	respected
Denotation of <i>Alice</i>	a — simple	$\lambda P.P(a)$ — baroque

And nothing is lost semantically. Applying the lifted name to a predicate β -reduces straight back:

$$(\lambda P. P(a))(\llbracket \text{sleeps} \rrbracket) \rightsquigarrow \llbracket \text{sleeps} \rrbracket(a)$$

The lift is a *repackaging*, not a change of meaning. That is what makes it a legitimate move rather than a fudge — and it is why **LIFT** and **LOWER** are inverse where both are defined.

Montague lifted everything. Most modern treatments lift only when forced, which is a practical compromise — and Problem Set 1 takes the simpler route.

Two Quantifiers, One Sentence

Every student read a book.

Surface ($\forall > \exists$) $\forall x[\text{student}(x) \rightarrow \exists y[\text{book}(y) \wedge \text{read}(x, y)]]$

each student read some book, possibly different ones

Inverse ($\exists > \forall$) $\exists y[\text{book}(y) \wedge \forall x[\text{student}(x) \rightarrow \text{read}(x, y)]]$

one particular book was read by every student

One string. One syntactic structure. Two meanings.

This is a crisis for compositionality as we stated it. If meaning is a function of structure, and there is one structure, where does the second meaning come from?

Why Ordinary Composition Gives Only One Reading

Build the sentence bottom-up with the rules we have.

- ① $a \text{ book} = \lambda P. \exists y[\text{book}(y) \wedge P(y)] \quad : \langle \langle e, t \rangle, t \rangle$
- ② The VP needs *read* applied to its object. But *read* wants type e , and *a book* is $\langle \langle e, t \rangle, t \rangle$.
Type mismatch.

The standard repair: let the object quantifier take scope over the VP.

$$\llbracket \text{read a book} \rrbracket = \lambda x. \exists y[\text{book}(y) \wedge \text{read}(x, y)]$$

Then the subject quantifier applies to that.

Result: $\forall x[\text{student}(x) \rightarrow \exists y[...]]$ — the **surface** reading.

Composition builds inside-out, so the deeper quantifier ends up with narrower scope. The inverse reading requires the *object* to somehow escape and take scope above the subject.

The Object-Position Problem

Subject quantifiers work because the NP applies to the VP. **Object quantifiers do not have that option.**

Alice read every book.

$\llbracket \text{read} \rrbracket : \langle e, \langle e, t \rangle \rangle$

wants an entity

$\llbracket \text{every book} \rrbracket : \langle \langle e, t \rangle, t \rangle$

is not an entity

The verb cannot apply to its own object.

This is the real source of the scope problem. Subject quantifiers are easy; object quantifiers are stuck. Every mechanism on the following slides exists to unstick them — and once a quantifier can move to fix a type mismatch, *how far* it moves becomes a question, and ambiguity follows.

Three Ways to Unstick It

Lift the verb	Give <i>read</i> the type $\langle\langle e, t \rangle, t \rangle, \langle e, t \rangle\rangle$ so it takes a quantifier directly. Works; every transitive verb gets a second, higher-typed entry.
Type-shift in place	Apply a general rule turning $\langle e, \langle e, t \rangle \rangle$ into the higher type. One rule instead of many entries — Hendriks' flexible types.
Move the quantifier	Let it take scope over the whole clause, binding a trace. QR, storage, continuations — the three mechanisms.

The first two fix the type mismatch but give only **one** reading each — whichever the rule builds in.

The third fixes the mismatch *and* predicts ambiguity, because a moved quantifier can land in more than one place. That is why it is the standard treatment, despite being the most machinery.

Three Mechanisms

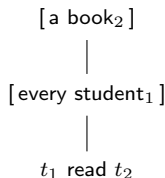
Mechanism	Idea	Tradition
Quantifier Raising	move the NP at LF, leave a trace	generative syntax
Cooper storage	set the quantifier aside, retrieve later	Montague grammar
Continuations	give the NP access to its own context	type-theoretic

All three do the same job: **let a quantifier take scope wider than its surface position**. They differ in where the machinery lives.

We will look at each briefly. None is *the* answer — this is a place where the field has several working proposals rather than one settled account, and it is worth seeing that clearly.

Mechanism 1: Quantifier Raising

Posit a level of **Logical Form** at which the quantified NP has moved, leaving a coindexed trace.



Surface order is *every student read a book*;
at LF the object has
raised *above* the subject.

Each raised NP binds its trace, and the LF is interpreted compositionally in the ordinary way.

Gain composition itself is untouched; ambiguity is *structural*

Cost an extra, unpronounced level of representation

The strategy: keep the semantics simple by letting the syntax be more complicated. Whether the extra level is real is a long-running debate in syntax.

Mechanism 2: Cooper Storage

Keep one level of structure. When you meet a quantified NP, **put it in a store** and leave a variable behind.

Step	Core meaning	Store
<i>read a book</i>	$\text{read}(x_1, y_2)$	$\{\langle \text{a book}, 2 \rangle\}$
+ <i>every student</i>	$\text{read}(x_1, y_2)$	$\{\langle \text{a book}, 2 \rangle, \langle \text{every student}, 1 \rangle\}$
retrieve	in either order	

The ambiguity comes from the order of retrieval. Retrieve the subject last and you get surface scope; retrieve the object last and you get inverse scope.

Gain one syntactic structure; ambiguity is *derivational*

Cost meanings are now pairs (core, store) — not just denotations

Mechanism 3: Continuations

Give each expression access to **its own surrounding context** — what happens to its value once it is computed.

Instead of $\llbracket \alpha \rrbracket$, we compute $\lambda k. \dots k(\llbracket \alpha \rrbracket) \dots$, where k is the continuation.

A quantifier is then an expression that *does something interesting* with its continuation:

$$\llbracket \text{a book} \rrbracket = \lambda k. \exists y [\text{book}(y) \wedge k(y)]$$

Scope ambiguity becomes the order in which continuations are composed.

Gain	one structure, one representation level, general mechanism
Cost	every type gets more complicated; hard to read

Note the family resemblance: a generalized quantifier $\lambda P. \dots P \dots$ already *is* a continuation-passing denotation. Type-lifting an NP and CPS-transforming it are the same operation. Barker (2002); Barker & Shan (2014).

Comparing the Three

	QR	Storage	Continuations
Extra representation level	yes	no	no
Ambiguity located in	structure	derivation	derivation
Meanings are	denotations	pairs	denotations
Generalizes beyond scope	no	partly	yes
Ease of implementation	moderate	easy	hard

What they share, and this is the point: every one of them adds machinery *beyond* function application. **Composition alone does not deliver scope ambiguity.**

Remember this when we reach Part IV. A proposal that “contraction implements function application” inherits exactly this problem. Getting application right is necessary and **not sufficient** — scope needs something more, in any framework.

Three Quantifiers, Six Readings

Every student showed a book to some teacher.

Three quantifiers means up to $3! = 6$ scope orderings:

$\forall > \exists_{bk} > \exists_{tc}$	each student, some book, some teacher
$\forall > \exists_{tc} > \exists_{bk}$	each student, some teacher, some book
$\exists_{bk} > \forall > \exists_{tc}$	one book, each student showed it to someone
$\exists_{bk} > \exists_{tc} > \forall$	one book, one teacher, all students
$\exists_{tc} > \forall > \exists_{bk}$	one teacher, each student showed some book
$\exists_{tc} > \exists_{bk} > \forall$	one teacher, one book, all students

In general, n quantifiers give up to $n!$ readings.

Two consequences. For **theory**: any mechanism must generate all and only the available readings. For **parsing**: the search space is factorial, which is why systems in practice generate one reading and move on.

Inverse Linking

Some scope facts are more surprising than others.

Someone from every city despises it.

The preferred reading is:

$$\forall x[\text{city}(x) \rightarrow \exists y[\text{from}(y, x) \wedge \text{despise}(y, x)]]$$

The embedded quantifier every city takes scope over the NP containing it — and binds the pronoun *it* in the object position.

This is called **inverse linking**, and it is hard for every mechanism. The quantifier must escape its own NP, outscope the NP it is embedded in, and still be available to bind a pronoun downstream. Any account that gets the simple cases right still owes an explanation here.

Practice: Which Reading?

For each sentence, give the logical form of the reading indicated.

- ① *Every student read a book* — inverse
- ② *Some student read every book* — surface
- ③ *No student read a book* — surface

Practice: Which Reading?

For each sentence, give the logical form of the reading indicated.

- ① *Every student read a book* — inverse
- ② *Some student read every book* — surface
- ③ *No student read a book* — surface

Answers.

- ① $\exists y[\text{book}(y) \wedge \forall x[\text{student}(x) \rightarrow \text{read}(x, y)]]$
- ② $\exists x[\text{student}(x) \wedge \forall y[\text{book}(y) \rightarrow \text{read}(x, y)]]$
- ③ $\neg \exists x[\text{student}(x) \wedge \exists y[\text{book}(y) \wedge \text{read}(x, y)]]$

Item 3 is worth a second look: with *no* in subject position, the two scope orderings happen to be *equivalent*. Not every sentence with two quantifiers is genuinely ambiguous — some ambiguities collapse, and which ones do is determined by the monotonicity properties of the determiners involved.

The Entailment Between Readings

inverse $\models$ surface

surface $\not\models$ inverse

If one book was read by every student, then certainly every student read some book. The converse fails.

Proof sketch of the entailment. Assume $\exists y[\text{book}(y) \wedge \forall x[\dots \rightarrow \text{read}(x, y)]]$. Fix such a y_0 . For any student x , we have $\text{read}(x, y_0)$ and $\text{book}(y_0)$, so $\exists y[\text{book}(y) \wedge \text{read}(x, y)]$ holds. Hence the surface reading. $\square$

This is the $\exists\forall \models \forall\exists$ result from Session 3, with the restrictor carried along. The counter-model for the converse is on the Session 3 slide.

Problem Set 1's writeup asks you for exactly this argument — and specifically whether it holds in general or only in our model.

Not Every Reading Is Available

Scope is not free. Some configurations are systematically unavailable.

Constraint	Example	Effect
Clause boundedness	<i>Alice thinks every student left</i>	$\forall$ tends to stay in its clause
Islands	<i>a man who saw every boy left</i>	$\forall$ cannot escape the relative clause
Preference	<i>every student read a book</i>	surface reading strongly preferred

A theory that generates all scope permutations overgenerates. The mechanisms on the previous slides must be constrained, and how to constrain them is an active research question — not a solved problem with an implementation detail left over.

This is also why real semantic parsers often generate only surface scope: the inverse reading is rarer, harder to constrain, and for most applications not worth the search-space cost. We saw that decision made explicitly in the λ -DCS lecture.

What This Means for Problem Set 1

Part E asks for both readings. You now have everything you need.

```
surface :: T
surface = every student (\x -> some' book (\y -> read' y x))

inverse :: T
inverse = some' book (\y -> every student (\x -> read' y x))
```

Notice what the scaffold quietly does for you. Haskell's lambda gives you variable binding for free, so you can nest the quantifiers in either order and the variables sort themselves out.

You are not implementing QR, storage, or continuations. You are writing both logical forms directly — which sidesteps the mechanism question entirely.

That is a legitimate simplification for a problem set, and it is worth knowing that it *is* a simplification. A real parser has to *derive* both from one syntactic structure. You are being handed the destination without the journey.

Summary

Concept	Key point
Generalized quantifier	NP of type $\langle\langle e, t \rangle, t\rangle$; a set of sets
Determiner	a relation between restrictor and scope
Conservativity	$D(Q)(P) \iff D(Q)(Q \cap P)$; a semantic universal
<i>only</i>	not conservative, therefore not a determiner
Monotonicity	$\text{MON}\uparrow$ / $\text{MON}\downarrow$; drives NPI licensing
LIFT	$x \mapsto \lambda P.P(x)$; makes all NPs one type
Coordination	the argument for uniform lifting
Scope ambiguity	one structure, two readings
Three mechanisms	QR, storage, continuations
The common point	all add machinery beyond function application
Entailment	inverse $\models$ surface, one direction only
Constraints	islands and clause boundedness; a theory can overgenerate

Coming Up

Session 6: Beyond the sentence

- Modality and intensionality — where extensions are not enough
- Dynamic semantics: meaning as context change
- What a sentence-level theory leaves out

Reading: EU 7.1–7.4. The SEP entry on generalized quantifiers is optional and good.

Then, closing Part I:

- **6A:** λ -DCS — the logical tradition at Freebase scale
- **6B:** semantic types as vectors — the bridge into Part II

Session 6B will show you that conservativity, which looked like a descriptive curiosity today, is what makes quantification representable in bounded space at all. Today's universals turn out to be a compression theorem.

Questions?

Next: Session 6 — Beyond the Sentence