More Theories, Formal semantics

Jirka Hana

Parts are based on slides by Carl Pollard

Charles University, 2011-11-12

Optimality Theory

- Universal set of violable constraints:
  - Faithfulness constraints:surface forms should be as close as to underlying forms
  - Markedness constraints: work on output (along the lines: CV structure is preferred, voiceless final sounds are preferred)
- Language differ in constraint rankings
- Language acquisition = discovering the ranking
- Mostly in phonology

HPSG

- The most widely used grammar framework in computational linguistics
- Fully formalized
- Model theoretic approach
- Objects: Typed feature structures - directed graph with labeled edges and nodes
- Grammar: set of constraints (a la Prolog + types + negation)
- Constraints can be expressed as AVMs

Expression, Meaning, and Reference

- Following Frege (1892), semanticists distinguish between the **meaning** (or **sense**) of a linguistic expression and its **reference** (or **denotation**).

Expression, Meaning, and Reference

- Following Frege (1892), semanticists distinguish between the **meaning** (or **sense**) of a linguistic expression and its **reference** (or **denotation**).
- We say an expression **expresses** its meaning, and **refers to**, or **denotes**, its reference.

Expression, Meaning, and Reference

- Following Frege (1892), semanticists distinguish between the **meaning** (or **sense**) of a linguistic expression and its **reference** (or **denotation**).
- We say an expression **expresses** its meaning, and **refers to**, or **denotes**, its reference.
- In general, the reference of an expression can be **contingent** (depend on how things are), while the meaning is independent of how things are (examples coming soon).

Expression, Meaning, and Reference

- Following Frege (1892), semanticists distinguish between the **meaning** (or **sense**) of a linguistic expression and its **reference** (or **denotation**).
- We say an expression **expresses** its meaning, and **refers to**, or **denotes**, its reference.
- In general, the reference of an expression can be **contingent** (depend on how things are), while the meaning is independent of how things are (examples coming soon).

Note: Here, we are ignoring the distinction between an *expression* and an *utterance of* an expression.

Examples

- The meaning of a declarative sentence is a **proposition**, while its reference is the truth value of that proposition.

Examples

- The meaning of a declarative sentence is a **proposition**, while its reference is the truth value of that proposition.
- The meaning of a common noun (e.g. *donkey*) or an intransitive verb (e.g. *brays*), is a **property**, while its reference is the set of things that have that property.

Examples

- The meaning of a declarative sentence is a **proposition**, while its reference is the truth value of that proposition.
- The meaning of a common noun (e.g. *donkey*) or an intransitive verb (e.g. *brays*), is a **property**, while its reference is the set of things that have that property.
- Names are controversial! Vastly oversimplifying:
    - **Descriptivism** (Frege, Russell) the meaning of a name is a **description** associated with the name by speakers; the reference is what satisfies the description.

Examples

- The meaning of a declarative sentence is a **proposition**, while its reference is the truth value of that proposition.
- The meaning of a common noun (e.g. *donkey*) or an intransitive verb (e.g. *brays*), is a **property**, while its reference is the set of things that have that property.
- Names are controversial! Vastly oversimplifying:
  - **Descriptivism** (Frege, Russell) the meaning of a name is a **description** associated with the name by speakers; the reference is what satisfies the description.
  - **Direct Reference Theory** (Mill, Kripke) the meaning of a name **is** its reference, so names are **rigid** (their reference is independent of how things are.)

Grammar and Meaning

- The grammar of a language specifies meanings of expressions.

Grammar and Meaning

- The grammar of a language specifies meanings of expressions.
- Grammar says nothing about reference.

Compositionality

The Principle of Compositionality: The meaning of an expression is a function of the meanings of its parts and of the way they are syntactically combined.

Compositionality

The Principle of Compositionality: The meaning of an expression is a function of the meanings of its parts and of the way they are syntactically combined.

A grammar specifies

- the meaning of words (or morphemes)
- how to derive a meaning of a complex expression from its components

Entailment

- *John ate a cake.*
- *A cake was eaten.*
- *There was a cake.*

Entailment

- *John ate a cake.*
- *A cake was eaten.*
- *There was a cake.*

Entailment: $\phi \models \psi$ iff (if ϕ is true then ψ must be true)

A Theory of Meanings and Extensions

Our theory will use the following sets as building blocks:

Prop	The **propositions** (sentence meanings)

A Theory of Meanings and Extensions

Our theory will use the following sets as building blocks:

Prop	The **propositions** (sentence meanings)
Bool	The **truth values** (extensions of propositions)

A Theory of Meanings and Extensions

Our theory will use the following sets as building blocks:

Prop	The **propositions** (sentence meanings)
Bool	The **truth values** (extensions of propositions)
Ind	The **individuals** (meanings of names).

A Theory of Meanings and Extensions

Our theory will use the following sets as building blocks:

Prop	The **propositions** (sentence meanings)
Bool	The **truth values** (extensions of propositions)
Ind	The **individuals** (meanings of names).
World	The **worlds** (ultrafilters of propositions)

A Theory of Meanings and Extensions

Our theory will use the following sets as building blocks:

Prop	The **propositions** (sentence meanings)
Bool	The **truth values** (extensions of propositions)
Ind	The **individuals** (meanings of names).
World	The **worlds** (ultrafilters of propositions)
One	The **unit set** $\{0\}$. It's conventional to call the member of this set $*$, rather than 0, since the important thing about it is that it is a singleton and not what its member is.

Propositions

- Propositions are primitive notions.

Propositions

- Propositions are primitive notions.
- We are agnostic only about their formal nature not about their properties.

The set of propositions (**Prop**) forms a pre-lattice:

The set of propositions (**Prop**) forms a pre-lattice:

- They are related by entailment:
  $\models$: $\mathbf{Prop} \times \mathbf{Prop} \rightarrow \mathbf{Bool}$

The set of propositions (**Prop**) forms a pre-lattice:

- They are related by entailment:
  $\models$: $\mathbf{Prop} \times \mathbf{Prop} \to \mathbf{Bool}$
- And the induced equivalence:
  $\equiv$: $\mathbf{Prop} \times \mathbf{Prop} \to \mathbf{Bool}$

The set of propositions (**Prop**) forms a pre-lattice:

- They are related by entailment:
  $\models$: **Prop** $\times$ **Prop** $\rightarrow$ **Bool**
- And the induced equivalence:
  $\equiv$: **Prop** $\times$ **Prop** $\rightarrow$ **Bool**
- Entailment is constrained to be a preorder (i.e., reflexive, transitive, but not antisymmetric)
  The absence of antisymmetry allows two propositions to entail each other and still be distinct objects. Equality implies equivalence but not vice versa.

The set of propositions (**Prop**) forms a pre-lattice:

- They are related by entailment:
  $\models$: $\mathbf{Prop} \times \mathbf{Prop} \rightarrow \mathbf{Bool}$
- And the induced equivalence:
  $\equiv$: $\mathbf{Prop} \times \mathbf{Prop} \rightarrow \mathbf{Bool}$
- Entailment is constrained to be a preorder (i.e., reflexive, transitive, but not antisymmetric)
  The absence of antisymmetry allows two propositions to entail each other and still be distinct objects. Equality implies equivalence but not vice versa.
- There are the usual glb/lub, top/bottom, complement, residual operations.

(Hyper)intensional types

Meaning, the kind of hyperintensional types is defined as follows:

- $\mathbf{Prop}, \mathbf{Ind}, \mathbf{One} \in \mathbf{Meaning}$.
- If $A, B \in \mathbf{Meaning}$ then
  - $A \times B \in \mathbf{Meaning}$.
  - $A \to B \in \mathbf{Meaning}$
- Nothing else is a hyperintensional type.

Examples of word meanings

syntax	semantics
proper name **Chiquita**	**Chiquita' : Ind**
common noun **donkey**	**donkey':Ind → Prop**
sentential adverb **obviously**	**obvious':Prop → Prop**
dummy pronoun $\mathbf{it}_d$	$* \in$ **One**
It is obvious that ...	

References

Meanings can be mapped to extensions (references).

References

Meanings can be mapped to extensions (references).

meaning type	maps to	reference type
Ind		**Ind**
Prop		**Bool**
Prop → **Prop**		**Prop** → **Bool**
etc.		

References

Meanings can be mapped to extensions (references).

meaning type	maps to	reference type
Ind		**Ind**
Prop		**Bool**
Prop $\rightarrow$ **Prop**		**Prop** $\rightarrow$ **Bool**
etc.		

meaning	reference
Chiquita': **Ind**	**Chiquita'**: **Ind**
$*$: **One**	$*$: **One**
donkey' : **Ind** $\rightarrow$ **Prop**	f : **Ind** $\rightarrow$ **Bool**; $f(i) \Leftrightarrow (i$ is a donkey$)$
obviously : **Prop** $\rightarrow$ **Prop**	g : **Prop** $\rightarrow$ **Bool**; $g(p) \Leftrightarrow (p$ is obvious$)$

Categorial Grammar

- logical formalism, several implications (usually written as / and \)
- small number of language independent rules (e.g., modus ponens = function application), the rest of grammar is in the lexicon (radical lexicalism)
- syntactic structure is an equivalence set of proofs
- Usually, surface form and semantics are derived in parallel.

Categorial Grammar

- we can say that verbs are functions taking noun phrases as their arguments
- $\dfrac{\text{john:NP} \quad \text{sleeps:NP}\backslash\text{S}}{\text{john sleeps:S}}$ application

  If there is an object john of type NP and an object sleeps of type NP\S then there is an object john sleeps of type S. Note that john sleeps is a syntactical object, not the actual surface form. We could have written x123 for john.

Recall

currying – transformation of a function with multiple parameters into a function taking a single argument (the first of the arguments of the original function) and returning a new function which takes the remainder of the arguments.

Recall

currying – transformation of a function with multiple parameters into a function taking a single argument (the first of the arguments of the original function) and returning a new function which takes the remainder of the arguments.

uncurried	curried
plus : (Int $\times$ Int) $\rightarrow$ Int	plus' : Int $\rightarrow$ (Int $\rightarrow$ Int)
plus$(3, 4)$	plus'(3)(4)

Recall

currying – transformation of a function with multiple parameters into a function taking a single argument (the first of the arguments of the original function) and returning a new function which takes the remainder of the arguments.

uncurried	curried
plus : (Int $\times$ Int) $\to$ Int	plus' : Int $\to$ (Int $\to$ Int)
plus(3, 4)	plus'(3)(4)

inc = plus'(1)

All together now

- Words, phrases and sentences are modeled as signs.
- Signs are the combinations of phonological, syntactic, and semantic objects that makes sense.
- The phonological component of a sign is the pronunciation of the syntactic component and the semantic component is the meaning of it.
- The type Sign is a subtype of the following tuple type
$$\begin{bmatrix} \mathit{phon} & \mathbf{Phon}^* \\ \mathit{syn} & \mathbf{Syn} \\ \mathit{sem} & \mathbf{Meaning} \end{bmatrix}$$
- **Syn** is the kind of syntactic types, i.e., set of basic types (**NP**, **N**, **S**, ...) closed under syntactic constructors ($\times$, $\Rightarrow$, ...).

All together now – Lexicon

- Lexicon consists of axioms of the form:

$$\vdash \begin{bmatrix} phon & [\text{bo}\text{ɪ}] : \mathbf{Phon}^* \\ syn & boy : \mathbf{N} \\ sem & boy' : \mathbf{Ind} \Rightarrow \mathbf{Prop} \end{bmatrix}$$

All together now – Lexicon

- Lexicon consists of axioms of the form:

$$\vdash \begin{bmatrix} \mathit{phon} & [\text{bo}\text{ɪ}] : \mathbf{Phon}^* \\ \mathit{syn} & \mathit{boy} : \mathbf{N} \\ \mathit{sem} & \mathit{boy}' : \mathbf{Ind} \Rightarrow \mathbf{Prop} \end{bmatrix}$$

- In abbreviated form:

[boɪ]: $\mathbf{Phon}^*$ boy: $\mathbf{N}$ boy': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$

All together now – Lexicon

- Lexicon consists of axioms of the form:

$$\vdash \begin{bmatrix} \textit{phon} & [\text{boɪ}] : \mathbf{Phon}^* \\ \textit{syn} & \textit{boy} : \mathbf{N} \\ \textit{sem} & \textit{boy}' : \mathbf{Ind} \Rightarrow \mathbf{Prop} \end{bmatrix}$$

- In abbreviated form:

[boɪ]: $\mathbf{Phon}^*$	boy: $\mathbf{N}$	boy': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[snoːrz]: $\mathbf{Phon}^*$	snores: $\mathbf{NP} \Rightarrow \mathbf{S}$	snore': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[sliːps]: $\mathbf{Phon}^*$	sleeps: $\mathbf{NP} \Rightarrow \mathbf{S}$	sleep': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$

All together now – Lexicon

- Lexicon consists of axioms of the form:

$$\vdash \begin{bmatrix} \textit{phon} & [\text{bo}ɪ] : \mathbf{Phon}^* \\ \textit{syn} & \textit{boy} : \mathbf{N} \\ \textit{sem} & \textit{boy}' : \mathbf{Ind} \Rightarrow \mathbf{Prop} \end{bmatrix}$$

- In abbreviated form:

[boɪ]: $\mathbf{Phon}^*$	boy: $\mathbf{N}$	boy': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[snoːrz]: $\mathbf{Phon}^*$	snores: $\mathbf{NP} \Rightarrow \mathbf{S}$	snore': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[sliːps]: $\mathbf{Phon}^*$	sleeps: $\mathbf{NP} \Rightarrow \mathbf{S}$	sleep': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[laʊdlɪ]: $\mathbf{Phon}^*$	loudly: $\mathbf{VP} \Rightarrow \mathbf{VP}$	loud': $(\mathbf{Ind} \Rightarrow \mathbf{Prop}) \Rightarrow (\mathbf{Ind} \Rightarrow \mathbf{Prop})$

All together now – Lexicon

- Lexicon consists of axioms of the form:

$$\vdash \begin{bmatrix} \textit{phon} & [\text{boɪ}] : \mathbf{Phon}^* \\ \textit{syn} & \textit{boy} : \mathbf{N} \\ \textit{sem} & \textit{boy}' : \mathbf{Ind} \Rightarrow \mathbf{Prop} \end{bmatrix}$$

- In abbreviated form:

[boɪ]: $\mathbf{Phon}^*$	boy: $\mathbf{N}$	boy': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[snoːrz]: $\mathbf{Phon}^*$	snores: $\mathbf{NP} \Rightarrow \mathbf{S}$	snore': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[sliːps]: $\mathbf{Phon}^*$	sleeps: $\mathbf{NP} \Rightarrow \mathbf{S}$	sleep': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[laʊdlɪ]: $\mathbf{Phon}^*$	loudly: $\mathbf{VP} \Rightarrow \mathbf{VP}$	loud': $(\mathbf{Ind} \Rightarrow \mathbf{Prop}) \Rightarrow (\mathbf{Ind} \Rightarrow \mathbf{Prop})$
[ɛvrɪ]: $\mathbf{Phon}^*$	every: $\mathbf{N} \Rightarrow \mathbf{NP}$	every' $= \lambda q, p.\lambda x.(q(x) \Rightarrow p(x))$: $(\mathbf{Ind} \Rightarrow \mathbf{Prop}) \times (\mathbf{Ind} \Rightarrow \mathbf{Prop}) \Rightarrow \mathbf{Prop}$

All together now – Lexicon

- Lexicon consists of axioms of the form:

$$\vdash \begin{bmatrix} \textit{phon} & [\text{boɪ}] : \mathbf{Phon}^* \\ \textit{syn} & \textit{boy} : \mathbf{N} \\ \textit{sem} & \textit{boy}' : \mathbf{Ind} \Rightarrow \mathbf{Prop} \end{bmatrix}$$

- In abbreviated form:

[boɪ]: $\mathbf{Phon}^*$	boy: $\mathbf{N}$	boy': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[snoːrz]: $\mathbf{Phon}^*$	snores: $\mathbf{NP} \Rightarrow \mathbf{S}$	snore': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[sliːps]: $\mathbf{Phon}^*$	sleeps: $\mathbf{NP} \Rightarrow \mathbf{S}$	sleep': $\mathbf{Ind} \Rightarrow \mathbf{Prop}$
[laʊdlɪ]: $\mathbf{Phon}^*$	loudly: $\mathbf{VP} \Rightarrow \mathbf{VP}$	loud': $(\mathbf{Ind} \Rightarrow \mathbf{Prop}) \Rightarrow (\mathbf{Ind} \Rightarrow \mathbf{Prop})$
[ɛvrɪ]: $\mathbf{Phon}^*$	every: $\mathbf{N} \Rightarrow \mathbf{NP}$	every' $= \lambda q, p.\lambda x.(q(x) \Rightarrow p(x))$: $(\mathbf{Ind} \Rightarrow \mathbf{Prop}) \times (\mathbf{Ind} \Rightarrow \mathbf{Prop}) \Rightarrow \mathbf{Prop}$
[ænd]: $\mathbf{Phon}^*$	and: $\forall A.A \times A \Rightarrow A$	and': $\forall A.A \times A \Rightarrow A$

All together now – Grammar

- function application in syntax corresponds to:
  - function application in semantics
  - concatenation in phonology
- Possibly other rules in individual sub-grammars. For example, phonotactic constraints in phonology.

every boy:

every boy:

Relevant lexicon:

$$\vdash \begin{bmatrix} \textit{phon} & [\text{bɔɪ}] : \textbf{Phon}^* \\ \textit{syn} & \textit{boy} : \textbf{N} \\ \textit{sem} & \textit{boy}' : \textbf{Ind} \Rightarrow \textbf{Prop} \end{bmatrix} \quad \vdash \begin{bmatrix} \textit{phon} & [\text{ɛvrɪ}] : \textbf{Phon}^* \\ \textit{syn} & \textit{every} : \textbf{N} \Rightarrow \textbf{NP} \\ \textit{sem} & \textit{every}' = \lambda q, p.\lambda x.(q(x) \Rightarrow p(x)) : \\ \multicolumn{2}{l}{(\textbf{Ind} \Rightarrow \textbf{Prop}) \times (\textbf{Ind} \Rightarrow \textbf{Prop}) \Rightarrow \textbf{Prop}} \end{bmatrix}$$

every boy:

Relevant lexicon:

$\vdash \begin{bmatrix} \textit{phon} & [\text{bɔɪ}] : \textbf{Phon}^* \\ \textit{syn} & \textit{boy} : \textbf{N} \\ \textit{sem} & \textit{boy}' : \textbf{Ind} \Rightarrow \textbf{Prop} \end{bmatrix}$ $\vdash \begin{bmatrix} \textit{phon} & [\text{ɛvrɪ}] : \textbf{Phon}^* \\ \textit{syn} & \textit{every} : \textbf{N} \Rightarrow \textbf{NP} \\ \textit{sem} & \textit{every}' = \lambda q, p.\lambda x.(q(x) \Rightarrow p(x)) : \\ (\textbf{Ind} \Rightarrow \textbf{Prop}) \times (\textbf{Ind} \Rightarrow \textbf{Prop}) \Rightarrow \textbf{Prop} \end{bmatrix}$

$$\vdash \begin{bmatrix} \textit{phon} & [\text{ɛvrɪ bɔɪ}] : \textbf{Phon}^* \\ \textit{syn} & \text{every(boy)} : \textbf{NP} \\ \textit{sem} & \text{every'(boy')} : (\textbf{Ind} \Rightarrow \textbf{Prop}) \Rightarrow \textbf{Prop} \end{bmatrix}$$

every boy:

Relevant lexicon:

$\vdash \begin{bmatrix} phon & [\text{bɔɪ}] : \textbf{Phon}^* \\ syn & boy : \textbf{N} \\ sem & boy' : \textbf{Ind} \Rightarrow \textbf{Prop} \end{bmatrix}$ $\vdash \begin{bmatrix} phon & [\text{ɛvrɪ}] : \textbf{Phon}^* \\ syn & every : \textbf{N} \Rightarrow \textbf{NP} \\ sem & every' = \lambda q, p.\lambda x.(q(x) \Rightarrow p(x)) : \\ (\textbf{Ind} \Rightarrow \textbf{Prop}) \times (\textbf{Ind} \Rightarrow \textbf{Prop}) \Rightarrow \textbf{Prop} \end{bmatrix}$

$$\vdash \begin{bmatrix} phon & [\text{ɛvrɪ bɔɪ}] : \textbf{Phon}^* \\ syn & \text{every(boy)} : \textbf{NP} \\ sem & \text{every'(boy')} : (\textbf{Ind} \Rightarrow \textbf{Prop}) \Rightarrow \textbf{Prop} \end{bmatrix}$$

every'(boy') =
$[\lambda q, p \, . \, \lambda x.(q(x) \Rightarrow p(x))](\lambda x \, . \, \text{boy'}(x))$

every boy:

Relevant lexicon:

$$\vdash \begin{bmatrix} \textit{phon} & [\text{bɔɪ}] : \textbf{Phon}^* \\ \textit{syn} & \textit{boy} : \textbf{N} \\ \textit{sem} & \textit{boy}' : \textbf{Ind} \Rightarrow \textbf{Prop} \end{bmatrix} \quad \vdash \begin{bmatrix} \textit{phon} & [\text{ɛvrɪ}] : \textbf{Phon}^* \\ \textit{syn} & \textit{every} : \textbf{N} \Rightarrow \textbf{NP} \\ \textit{sem} & \textit{every}' = \lambda q, p.\lambda x.(q(x) \Rightarrow p(x)) : \\ (\textbf{Ind} \Rightarrow \textbf{Prop}) \times (\textbf{Ind} \Rightarrow \textbf{Prop}) \Rightarrow \textbf{Prop} & \end{bmatrix}$$

$$\vdash \begin{bmatrix} \textit{phon} & [\text{ɛvrɪ bɔɪ}] : \textbf{Phon}^* \\ \textit{syn} & \text{every(boy)} : \textbf{NP} \\ \textit{sem} & \text{every'(boy')} : (\textbf{Ind} \Rightarrow \textbf{Prop}) \Rightarrow \textbf{Prop} \end{bmatrix}$$

every'(boy') =

$$[\lambda q, p \ . \ \lambda x.(q(x) \Rightarrow p(x))](\lambda x \ . \ \text{boy'}(x))$$
$$\lambda p \ . \ \lambda x(\text{boy'}(x) \Rightarrow p(x))$$

every boy sleeps and snores loudly

every boy sleeps and snores loudly

$$\vdash \begin{bmatrix} \textit{phon} & [\text{ɛvrɪ bɔɪ sliːps ænd snoːrz laʊdlɪ}] : \textbf{Phon}^* \\ \textit{syn} & \text{and}(\text{sleeps}, \text{loud}(\text{snore}))(\text{every}(\text{boy})) : \textbf{S} \\ \textit{sem} & \text{and'}(\text{sleep'}, \text{loud'}(\text{snore'}))(\text{every'}(\text{boy'})) : \textbf{Prop} \end{bmatrix}$$

every boy sleeps and snores loudly

$$\vdash \left[\begin{array}{ll} \textit{phon} & [\text{ɛvrɪ bɔɪ sliːps ænd snoːrz laʊdlɪ}] : \mathbf{Phon}^{*} \\ \textit{syn} & \text{and}(\text{sleeps}, \text{loud}(\text{snore}))(\text{every}(\text{boy})) : \mathbf{S} \\ \textit{sem} & \text{and'}(\text{sleep'}, \text{loud'}(\text{snore'}))(\text{every'}(\text{boy'})) : \mathbf{Prop} \end{array}\right]$$

$$\text{and'}(\text{sleep'}, \text{loud'}(\text{snore'}))(\text{every'}(\text{boy'})) =$$
$$[\lambda p.\lambda x(\text{boy'}(x) \Rightarrow p(x))](\lambda s.\text{and'}(\text{sleep'}, \text{loudly'}(\text{snore'}))(s)) =$$

every boy sleeps and snores loudly

$$\vdash \left[\begin{array}{ll} \textit{phon} & [\text{ɛvrɪ bɔɪ sliːps ænd snoːrz laʊdlɪ}] : \mathbf{Phon}^* \\ \textit{syn} & \text{and}(\text{sleeps}, \text{loud}(\text{snore}))(\text{every}(\text{boy})) : \mathbf{S} \\ \textit{sem} & \text{and'}(\text{sleep'}, \text{loud'}(\text{snore'}))(\text{every'}(\text{boy'})) : \mathbf{Prop} \end{array} \right]$$

$$\begin{array}{l} \text{and'}(\text{sleep'}, \text{loud'}(\text{snore'}))(\text{every'}(\text{boy'})) = \\ \quad [\lambda p.\lambda x(\text{boy'}(x) \Rightarrow p(x))](\lambda s.\text{and'}(\text{sleep'}, \text{loudly'}(\text{snore'}))(s)) = \\ \quad \lambda x.(\text{boy'}(x) \Rightarrow \text{and'}(\text{sleep'}, \text{loud'}(\text{snore'}))(x)) \end{array}$$