L98: Introduction to Computational Semantics
Lecture 6: Truth

Weiwei Sun and Simone Teufel

Natural Language and Information Processing Research Group
Department of Computer Science and Technology
University of Cambridge

Lent 2021/22
Lecture 6: Truth
1. Ferdinand de Saussure
2. World model and discourse referents
3. Functions and λs
4. Truth conditions
5. First-Order Predicate Logic
Ferdinand de Saussure
De Saussure: The linguistic sign is a two-sided psychological entity:

- **signifier**: ‘sound-image’
- **signified**: ‘concept’
Example: 止戈为武

- 止 initially meant foot, walk, go
- 戈 is an old-fashioned weapon
- 武 means military
- When 武 was created, the meaning of was “take your weapon and walk, go to war”
- So initially, there is semantic compositionality
- After many years, the meanings of the parts shifted and people could no longer see the compositionality.
- 止 now means “stop”.
- Now some philosophers reinterpret 武 as: Stop using weapons; that is what a military should do.
Arbitrariness of the sign

• De Saussure stated that the link between the signified and signifier is *arbitrary*.
• The example of 武 shows that it does not matter if language users know the complicated (non-arbitrary) history of this sign.
• The connection has become arbitrary.
• All you need to know to communicate is to know that 武 means military.
• Triumph of arbitrariness of the sign.
Old: mind and language

Mind

Language
Reality, mind and language

world: an animal

speaker’s thought

black swan

speaker’s words

signified

signifier

listener’s thought
Reality, mind and language

world: Trump elected

speaker’s thought

signifier

a black swan event

signified

signifier

listener’s thought
Reality, mind and language

world: Trump elected

a black swan event
speaker’s words
Natural Language Understanding

Example: Visual QA

Something gets lost if a system goes directly from words to images (diagonal)

- Idea of such a system is that the meaning of the language string creates something like a image in the mind.
- But we have just seen that it’s not a picture that is in the mind, otherwise the misunderstanding between listener and speaker would not have happened.
- This is why deep NLU needs to model the thought
- Rest of this lecture: how can we model the thought
World Model and Discourse Referent
Domains of interpretation

- the real world
- a part of the real world
- a hypothesized model of the real world
  - e.g. Shapeworld: objects with properties in positions
  - or something more complicated
- some constructed model in the case of an artificial language
Desired properties of a world model

A world model is an abstracted, simplified version of our world.

- The world model should be precise.
- Reflect the complexity of the phenomena we think are important.
- Drop other things we don’t care about.
- Should have a systematic way to be constructed.

- Its components should be transparent, i.e., it should be easy to see what in our world is what in the model.
- There should be a close link between our model and those phenomena in the real world we care about.
Our world model

It consists of

- **discourse referents.** unique variables standing in for actual people and objects in the world
- **semantic predicates.** functions representing “buckets” (certain nouns) and properties and events

We will start with proper names and simple predicates.

*Trump gave Johnson a golden lighter.*

The term “predicate” is also often used to describe a particular syntactic elements. We use “semantic predicate” to distinguish these two concepts.
Extension and intension

The extension of a linguistic expression is the set of things it extends to, or applies to.

Example: *politician*  
{Trump, Johnson, ...}
Extension and intension

The extension of a linguistic expression is the set of things it extends to, or applies to.

Example: *politician*

\{Trump, Johnson, \ldots\}

The above set could also be the extension of *lier*. 
Extension and intension

The extension of a linguistic expression is the set of things it extends to, or applies to.

Example: politician

\{Trump, Johnson, \ldots\}

The above set could also be the extension of liar.

Intensional semantics

e.g. a description from wikipedia

A politician is a person active in party politics, or a person holding or seeking an elected seat in government.

• intensional, but imprecise description
Discourse referents

World Model

Language

“Boris Johnson”

“Angela Merkel”

“Donald Trump”

“politician”

“lighter”

“silver”

“golden”
Discourse referents

**World Model**

- i56
- i45
- i34
- i23
- i456
- i345
- i234
- i123
- i12

**Language**

- “Boris Johnson”
- “Angela Merkel”
- “Donald Trump”
- “politician”
- “lighter”
- “silver”
- “golden”
Discourse referents

World Model

Language

“Boris Johnson”

“Angela Merkel”

“Donald Trump”

“politician”

“lighter”

“silver”

“golden”
Discourse referents

World Model

Language

“Boris Johnson”

“Angela Merkel”

“Donald Trump”

“politician”

“lighter”

“silver”

“golden”
Discourse referents

World Model

Language

“Boris Johnson”
“Angela Merkel”
“Donald Trump”
“politician”
“lighter”
“silver”
“golden”
Discourse referents

World Model

Language

“Boris Johnson”
“Angela Merkel”
“Donald Trump”
“politician”
“lighter”
“silver”
“golden”
“golden lighter”
Discourse referents

**World Model**

- i56
- i45
- i34
- i45
- i23
- i12
- i234
- i123
- i456
- i345

**Language**

- “Boris Johnson”
- “Angela Merkel”
- “Donald Trump”
- “politician”
- “lighter”
- “silver”
- “golden”
- …
Extensional interpretation

- An interpretation function \([J \, K]\) maps language expressions onto objects, sets of objects, sets of sets of..., of the world model.
  e.g. \([\text{politician}] = \{i_{12}, i_{23}, i_{34}, i_{45}, i_{56}\}\)
- In this lecture, objects of the world model are discourse referents.
- To simplify, proper names are mapped to \textbf{unique} discourse referents.
  e.g. \([\text{Angela Merkel}] = i_{34}\)
Functions and λs
Buckets/sets $\rightarrow$ functions

i56  i45  i34  i23

i12

i234  i123

i456  i345

1

0
Buckets/sets → functions

1

i56 → i23
i45 → i34
i34 → i23
i12

i234 → i123
i456 → i345

politician'(x)
Buckets/sets $\rightarrow$ functions

lighter'(x)
Buckets/sets → functions

silver'(x)
Buckets/sets $\rightarrow$ functions

\[
\begin{align*}
&i_{56} \quad i_{45} \quad i_{34} \quad i_{23} \quad i_{12} \\
i_{234} \quad i_{123} \quad i_{456} \quad i_{345}
\end{align*}
\]

\text{golden'}(x)
Predicates are functions; predicates are sets.

Q What is the meaning of *politician*?
A politician'

- politician’ is a semantic predicate which is a set and also a function.
- Discourse referents are mapped to either 0 or 1 through politician’. The referents mapped to 1 indicate politicians.
- It is a great idea to define functions with a minimal programming language — λ-calculus.
Building functions

$\lambda$-calculus — a simple notation for functions and application

- $\beta$-reduction/function application:

$$[\lambda x. M](N) \rightarrow M[x := N]$$

- Apply a $\lambda$-term to an argument, and get a value.

More online: https://plato.stanford.edu/entries/lambda-calculus/

Example

- $f(x) = x^2 \leftrightarrow [\lambda x. [x^2]]$
- $f(5) = 25 \leftrightarrow [\lambda x. [x^2]](5) = 25$
- $g(x, y) = x^2 + y^2 \leftrightarrow [\lambda x. [\lambda y. [x^2 + y^2]]]$
- $g(2, 1) = 5 \leftrightarrow [\lambda x. [\lambda y. [x^2 + y^2]]](2)(1) = 5$
Simple types

From a nonempty set $\text{BasTyp}$ of basic types, the set $\text{Typ}$ is the smallest set such that

• $\text{BasTyp} \subseteq \text{Typ}$,

• $\langle \sigma, \tau \rangle \in \text{Typ}$ if $\sigma, \tau \in \text{Typ}$.

A type of the form $\langle \sigma, \tau \rangle$ is said to be a functional type.

Example

• Assume $e$ for individuals and $t$ for true/false,

• then $\langle e, t \rangle$ is the type for unary relations,

• and $\langle \langle e, t \rangle, \langle e, t \rangle \rangle$ is for the type of a function mapping unary relations into unary relations.
Simple types

From a nonempty set $\text{BasTyp}$ of *basic types*, the set $\text{Typ}$ is the smallest set such that

- $\text{BasTyp} \subseteq \text{Typ}$,
- $\langle \sigma, \tau \rangle \in \text{Typ}$ if $\sigma, \tau \in \text{Typ}$.

A type of the form $\langle \sigma, \tau \rangle$ is said to be a *functional type*.

**Example**

- Assume $e$ for individuals and $t$ for true/false,
- then $\langle e, t \rangle$ is the type for unary relations,
- and $\langle \langle e, t \rangle, \langle e, t \rangle \rangle$ is for the type of a function mapping unary relations into unary relations.
e, t and e to t

Gottlob Frege

There are only two atomic things, truth values and individuals. All other things are created by function application.

entity e

truth value t

1

0
e, t and e to t

Gottlob Frege

There are only two atomic things, truth values and individuals. All other things are created by function application.

entity e

truth value t

\[ \lambda x.\text{politician}'(x) : \langle e, t \rangle \]
\[ \lambda x.\text{lighter}'(x) : \langle e, t \rangle \]
\[ \lambda x.\text{silver}'(x) : \langle e, t \rangle \]
\[ \lambda x.\text{golden}'(x) : \langle e, t \rangle \]
Syntactico-semantic composition

\[ S \left[ VP \right] \left[ NP \right] = \text{smoke}'(i23) \]

\[ [NP] = i23 \]

\[ [VP] = \lambda x.\text{smoke}'(x) \]

\[ N \]

\[ [Johnson] = i23 \]

\[ [smokes] = \lambda x.\text{smoke}'(x) \]

\[ V \]

\[ \text{Johnson} \]

\[ \text{smokes} \]

\[ \lambda x.\text{smoke}'(x) \]

\[ \langle e, t \rangle \]

\[ [Trump] \]

\[ [Johnson] \]

\[ 1 \]

\[ 0 \]

\[ i123 \]

\[ i234 \]

\[ i345 \]

\[ i456 \]

\[ i56 \]

\[ i45 \]

\[ i34 \]
Syntactico-semantic composition

\[ S \left[ VP \left[ NP \right] \right] = \text{smoke}'(i23) \]

\[ \left[ VP \right] \left[ NP \right] = i23 \]

\[ \left[ NP \right] = i23 \]

\[ \left[ VP \right] = \lambda x.\text{smoke}'(x) \]

\[ \left[ NP \right] = \lambda x.\text{smoke}'(x) \]

\[ \left[ Johnson \right] = i23 \]

\[ \left[ Johnson \right] = i23 \]

\[ \left[ Johnson \right] = \text{smokes} \]

\[ \left[ smokes \right] = \lambda x.\text{smoke}'(x) \]

\[ \left[ Trump \right] = i123 \]

\[ \left[ Trump \right] = i123 \]

\[ \left[ e, t \right] \]

\[ \lambda x.\text{smoke}'(x) \]

\[ \left[ e, t \right] \]
Syntactico-semantic composition

\[ S \left[ VP \left[ NP \right] \right] = \text{smoke}'(i23) \]

\[ \text{NP} \left[ NP \right] = i23 \]

\[ \text{NP} \left[ Johnson \right] = i23 \]

\[ \text{VP} \left[ VP \right] = \lambda x.\text{smoke}'(x) \]

\[ \text{VP} \left[ \text{smokes} \right] = \lambda x.\text{smoke}'(x) \]

\[ \lambda x.\text{smoke}'(x) \langle e, t \rangle \]
Syntactico-semantic composition

\[
S \left[ VP \right] \left( \left[ NP \right] \right) = smoke'(i23)
\]

\[
NP \left[ NP \right] \left[ VP \right] = i23 = \lambda x. smoke'(x)
\]

\[
\frac{N}{Johnson} = i23 = \lambda x. smoke'(x)
\]

\[
\frac{V}{smokes} = i234 = \lambda x. smoke'(x)
\]

\[
\lambda x. smoke'(x) \langle e, t \rangle
\]

\[
[Johnson] \quad [Trump]
\]

\[
i56 \quad i45 \quad i34
\]

\[
i234 \quad i123 \quad i456 \quad i345
\]
Compositional semantics

• \([Johnson\ smokes]\) is not listed in the lexicon.
• But the interpretation of \(Johnson\ smokes\) can still be derived from its parts along with a syntactic analysis.
• Finite means make infinite interpretation possible.
• This is exactly the point of compositional semantics
• and note that we have remained precise
• This means we can use this thing we just built as a meaning representation of the kind we wanted in Lecture 1.
Transitive verbs

Johnson *kissed* Trump

\[
\lambda x. \left[ \lambda y. \text{kiss}'(y, x) \right]
\]

\[
\langle e, \langle e, t \rangle \rangle
\]

1

0
Transitive verbs

Johnson kissed Trump
Transitive verbs

\[ \lambda x. [\lambda y. \text{kiss'}(y, x)] (\langle e, \langle e, t \rangle \rangle) \]
Syntactico-semantic composition

\[ S \equiv \text{[VP]}([\text{NP}]) = \text{kiss'}(i_{23}, i_{12}) \]

\[ \text{NP} \equiv [\text{NP}] = i_{23} \]

\[ \text{VP} \equiv [\text{VP}] = \lambda y. \text{kiss'}(y, i_{12}) \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{VP} \equiv \lambda y. \text{kiss'}(y, i_{12}) \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]

\[ \text{NP} \equiv [\text{NP}] = i_{12} \]

\[ \text{V} \equiv [\text{V}] = \lambda x. [\lambda y. \text{kiss'}(y, x)] = i_{12} \]

\[ \text{N} \equiv [\text{N}] = i_{12} \]
Syntactico-semantic composition

\[ S \left[ VP \right] \left[ NP \right] = \text{kiss'}(i_{23}, i_{12}) \]

\[ \text{NP} \left[ NP \right] = i_{23} \]
\[ \text{VP} \left[ VP \right] = \lambda y. \text{kiss'}(y, i_{12}) \]

\[ \text{N} \left[ N \right] = i_{23} \]
\[ \text{V} \left[ V \right] = \lambda x. \left[ \lambda y. \text{kiss'}(y, x) \right] \]

\[ \text{NP} \left[ NP \right] = i_{12} \]

\[ \text{N} \left[ N \right] = i_{12} \]
\[ \text{Trump} \]

Lexicon Look up

Lexicon Look up

Lexicon Look up

The type of this VP is the same to the intransitive verb.

The type of this V can be inferred from the type of the result, i.e. VP, and the type of the argument, i.e. the object.
Syntactico-semantic composition

\[ S = \left[ VP \right] \left[ \left[ NP \right] \right] = \text{kiss'}(i_{23}, i_{12}) \]

\[ \left[ VP \right] = \lambda y. \text{kiss'}(y, i_{12}) \]

\[ \left[ V \right] = \lambda x. [\lambda y. \text{kiss'}(y, x)] \]

\[ \left[ NP \right] = i_{12} \]

\[ \left[ N \right] = i_{12} \]

\[ \text{Johnson} \]

\[ \text{kissed} \]

\[ \text{Trump} \]
Syntactico-semantic composition

\[ \text{S} \left[ \text{VP} \left( \left[ \text{NP} \right] \right) \right] = \text{kiss'}(i23, i12) \]

NP  
\[ \left[ \text{NP} \right] = i23 \]

N   
\[ \left[ \text{N} \right] = i23 \]

VP  
\[ \left[ \text{VP} \right] = \lambda y. \text{kiss'}(y, i12) \]

V   
\[ \left[ \text{V} \right] = \lambda x. \left[ \lambda y. \text{kiss'}(y, x) \right] = i12 \]

NP  
\[ \left[ \text{NP} \right] = i12 \]

Function Application

\( \langle e, t \rangle \)

The type of this VP is the same to the intransitive verb.

Lexicon

Look up

Lexicon

Look up

Lexicon

Look up

Lexicon

Look up

Lexicon

Look up

Lexicon

Look up

Function Application
Syntactico-semantic composition

\[ S\left[ VP\right]\left[ NP\right] = \text{kiss'}(i23, i12) \]

The type of this VP is the same to the intransitive verb.

\[ \langle e, t \rangle \]

The type of this V can be inferred from the type of the result, i.e. VP, and the type of the argument, i.e. the object.

\[ \langle e, \langle e, t \rangle \rangle \]

Function Application

\[ \langle e, \langle e, t \rangle \rangle \]

The type of this VP

\[ = \lambda y.\text{kiss'}(y, i12) \]

\[ = \lambda x.\left[ \lambda y.\text{kiss'}(y, x) \right] = i12 \]

\[ \text{Johnson} \]

\[ \text{kissed} \]

\[ \text{Trump} \]

\[ \langle e, \text{Johnson} \rangle \]

\[ \text{kissed} \]

\[ \langle e, \text{Trump} \rangle \]
Syntactico-semantic composition

\[ S \left[ VP \right] \left[ \left[ NP \right] \right] = \text{kiss'}(i_{23}, i_{12}) \]

\[ \text{NP} \left[ \left[ NP \right] \right] = i_{23} \]

\[ \text{VP} \left[ \left[ VP \right] \right] = \lambda y. \text{kiss'}(y, i_{12}) \]

\[ \text{N} \left[ \left[ N \right] \right] = i_{23} \]

\[ \text{V} \left[ \left[ V \right] \right] = \lambda x. \left[ \lambda y. \text{kiss'}(y, x) \right] \]

\[ \text{NP} \left[ \left[ NP \right] \right] = i_{12} \]

\[ \text{N} \left[ \left[ N \right] \right] = i_{12} \]

\[ \text{Johnson} \]

\[ \text{kissed} \]

\[ \text{Trump} \]
What should we know for a lexical entry?

- **kissed**
- syntactic category: V
- semantic type: $\langle e, \langle e, t \rangle \rangle$
- semantic interpretation: $\lambda x. [\lambda y. \text{kiss}'(y, x)]$
Truth-Conditions
Meanings as truth conditions

Ludwig Wittgenstein

To know the meaning of a sentence is to know how the world would have to be for the sentence to be true.

The meaning of words and sentence parts is their contribution to the truth-conditions of the whole sentence.
The truth-conditional tradition

Consider three different word models: Different people smoke

Word Model 1

Word Model 2

Word Model 3
The truth-conditional tradition

Consider three different word models: Different people smoke

Word Model 1

Word Model 2

Word Model 3
The truth-conditional tradition

Consider three different word models: Different people smoke

Word Model 1

Word Model 2

Word Model 3
First-Order Predicate Logic (FOPL)
Davidsonian semantics: Adding event variables

What is the type of 

gives? 
e— individual.
Davidsonian semantics: Adding event variables

What is the type of \( J \) gives \( K \)?

\( e \) — individual.
What is the type of \([gives]\)? e — individual.
Ditransitive verb

T gives J a golden lighter

[a golden lighter] gives

Trump gives

Johnson
Ditransitive verb

What is the type of \([gives]\)? \(e\) — individual.
Neo-Davidsonian semantics: Further decomposition

[Agent] [Theme] [Recipient]

\[
T \text{ gives } J \text{ a golden lighter}
\]
Neo-Davidsonian semantics: Further decomposition

Further decomposition of the event structure

T gives J a golden lighter

Semantic roles:
AGENT  THEME  RECIPIENT
Lexicalised vs unlexicalised

Before Davidson

- \([gives]([Trump], [Johnson], [a golden lighter])\)
- \(\lambda x. [\lambda y. [\lambda z. give'(z, x, y)]]\)
- \(\langle e, \langle e, \langle e, t \rangle \rangle \rangle\)
- Lexicalised: the lexical entry contains rich information of arguments.

Davidsonian

- \([gives](e, [Trump], [Johnson], [a golden lighter])\)
- Lexicalised

Neo-Davidsonian

- \([gives](e) \land \text{AGENT}(e, [Trump]) \land \text{RECIPIENT}(e, [Johnson]) \land \text{THEME}(e, [a golden lighter])\)
- Modularisation of information
- Unlexicalised: the lexical entry doesn’t need to know argument structure.
First-order predicate logic

- What is \([\text{every student smokes}]\)?
  \[\forall x (\text{student}'(x) \rightarrow \text{smoke}'(x))\]

- What is \([\text{some students smoke}]\)?
  \[\exists x (\text{student}'(x) \land \text{smoke}'(x))\]
First-order predicate logic

• What is $\forall x (\text{student}'(x) \rightarrow \text{smoke}'(x))$?
• What is $\exists x (\text{student}'(x) \land \text{smoke}'(x))$?
First-order predicate logic

- What is $\forall x (\text{student}'(x) \rightarrow \text{smoke}'(x))$?
- What is $\exists x (\text{student}'(x) \land \text{smoke}'(x))$?

\[
\begin{align*}
\forall x (\text{student}'(x) \rightarrow \text{smoke}'(x)) \\
\exists x (\text{student}'(x) \land \text{smoke}'(x))
\end{align*}
\]
First-order predicate logic

- What is \([\text{every student smokes}]\)?
- What is \([\text{some students smoke}]\)?

\[ \forall x (\text{student}'(x) \rightarrow \text{smoke}'(x)) \]
\[ \exists x (\text{student}'(x) \land \text{smoke}'(x)) \]
First-order predicate logic

- What is \([\text{every student smokes}]\)?
  \[ \forall x (\text{student}'(x) \rightarrow \text{smoke}'(x)) \]
- What is \([\text{some students smoke}]\)?
  \[ \exists x (\text{student}'(x) \land \text{smoke}'(x)) \]
Truth of these statements in our world model?

In the world where *Trump gave Johnson a golden lighter* is true, which one of the following is true?

- Johnson gave Trump a lighter
- Trump gave Johnson a silver lighter
- Johnson was given a lighter
Readings