EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Philosophy
General Linguistics

Semantics & Pragmatics SoSe 2020

Lecture 9: Formal Semantics (Summary)

19/05/2020, Christian Bentz

Updated Schedule (2020)

21/04/2020	**Lecture 1**	Organization & Introduction
23/04/2020	**Lecture 2**	Information Theory I
28/04/2020	**Lecture 3**	Information Theory II
30/04/2020	**Lecture 4**	Formal Semantics I: Propositional Logic
05/05/2020	**Lecture 5**	Formal Semantics II: Predicate Logic
07/05/2020	**Lecture 6**	Formal Semantics III: Second-Order Logic
12/05/2020	**Lecture 7**	Formal Semantics IV: Type Theory
14/05/2020	**Lecture 8**	Formal Semantics V: Lambda Calculus
19/05/2020	**Lecture 9**	Summary: Formal Semantics
21/05/2020		**Ascension Day (Christi Himmelfahrt)**
26/05/2020	**Lecture 10**	Applications & Current Research
28/05/2020	**Lecture 11**	Further Topics in Semantics: Modality
		Pentecost Holidays (Pfingstferien)

Updated Schedule (2020)

09/06/2020	**Lecture 12**	Further Topics in Semantics: Evidentiality
11/06/2020		**Corpus Christi (Fronleichnam)**
16/06/2020	**Lecture 13**	Introduction Pragmatics
18/06/2020	**Lecture 14**	Discourse Representation Theory I
23/06/2020	**Lecture 15**	Discourse Representation Theory II
25/06/2020	**Lecture 16**	Implicatures
30/06/2020	**Lecture 17**	Presupposition I
02/07/2020	**Lecture 18**	Presupposition II
07/07/2020	**Lecture 19**	Speech Acts I
09/07/2020	**Lecture 20**	Speech Acts II
14/07/2020	**Lecture 21**	Conversational Structure
16/07/2020	**Lecture 22**	Pragmatic Universals
21/07/2020	**Lecture 23**	Summary: Pragmatics
23/07/2020	**Exam**	

Q&A

Lecture 8 (λ-calculus)

- *In the last example you provided on slide 32: shouldn't the lambda conversion step (right column) have the lambda operator around the quantified formula?* – Yes, since we said that lambda-conversion is not possible here, the lambda-operator should be left in place. I corrected this:
  $\forall X(X(a) \wedge X(b))(C)$ **x**
  has to be
  $\lambda X(\forall X(X(a) \wedge X(b)))(C)$ **x**

Q&A

Lecture 8 (λ-calculus)

- *In the first example of slide 35, there is a closing bracket missing. Would the missing bracket be before the lambda arguments(?) as in λx(λy(L(y)(x)))(j)(b), or should it reflect the scope of the lambda operator as in λx(λy(L(y)(x))(j))(b)?* – Yes, it has to be put as in the first suggestion.
- *On slide 13 example $\mathcal{C}$(B(j)(x)). Isn't this invalid since $\mathcal{C}$ takes an argument of type $\langle e, t \rangle$, while B(j)(x) is of type t?* – Yes, that's true. I corrected it to $\mathcal{C}$(B(j)).

Q&A

Lecture 8 (λ-calculus)

- *Slide 32, in particular concerning the λ-Abstraction expression $\lambda x(\exists xF(x) \rightarrow S(x))$. Conversion is here not possible since x is bound in one instance by $\exists$. However, if we change this to $\lambda x(\exists yF(y) \rightarrow S(x))$, then conversion of x would be possible. But aren't these two λ-expressions equivalent?* – It is correct that the latter expression can be converted, while the former cannot be converted. However, these two expressions cannot be seen as necessarily equivalent, since for variables in this logic language we cannot necessarily assume that x = y. This is pointed out by Gamut (1991, Volume 2, p. 109), where they say that, for instance, $\exists$yRxy cannot be assumed to necessarily be equivalent to $\exists$yRyy. This is because R could in fact represent the relation y $\neq$ x.

Q&A

Lecture 8 (λ-calculus)

- *You said in the lecture that the order of application for two-place predicates is such that we first combine the object of a transitive clause with the verb and then the subject, e.g. (L(m))(j) for "John loves Mary". Is this just a convention or is there psycholinguistic evidence for this?* – This is a syntactic convention. In some syntactic frameworks, binarization of tree structures is assumed. If we have binarized trees, then we have to decide which element combines first with the verb. The object is then mostly chosen because its case is directly determined by the verb. However, there is (as far as I know) no straightforward psycholinguistic (or other) evidence for these choices. For a discussion of the controversy about binarization see also Müller (2019), Grammatical theory, Chapter 18.

Q&A

Tutorial Week 3 Exercises

- The usage of the term “nephews” in “Charles and John are brothers or nephews” is misleading, since this suggests that Charles is a nephew of John and John is a nephew of Charles (just like for the relation of brothers). I replaced this with “Charles and John are brothers or cousins”. Note that then we also have to use exclusive or (XOR), since they cannot be both brothers and cousins. The translation is then Bcj XOR Ccj. There is an alternative (though rather unlikely) reading in which “brother” and “nephew” might be seen as one-place predicates (or two-place predicates where only one argument is specified and the other takes a variable), i.e. that Charles and John have the property of being brother or nephew (of some other persons, not of one another), in this case we could have: $(Bc \wedge Bj) \vee (Cc \wedge Cj)$.

Q&A

Tutorial Week 3 Exercises

- "All professional football players are ambitious" is now translated as: $\forall x((Px \wedge Fx) \rightarrow Ax)$. Note that translating "professional football player" as $Px \wedge Fx$ is somewhat problematic, since "x is professional" and "x is a football player" could also mean that x is professional in some other regard, not necessarily regarding being a football player. In fact, this is one of the reasons why we might want to go beyond predicate logic towards type theory. However, if we want to stick to predicate logic, and translate the adjective here separately, then this is the only way we can do it. An alternative is to just consider "professional football player" as one predicate, e.g. Fx.

Q&A

Tutorial Week 3 Exercises

- *To translate a sentence with everybody/everyone into predicate logic, e.g. "Everybody loves Mary", wouldn't we need to also define everybody/everyone as a person, i.e.* $\forall x(Px \rightarrow Lxm)$*?* – It is possible to do this to disambiguate between *everything* and *everybody* in the domain of discourse. However, Gamut (1991), for instance, do not require such disambiguation.

Overview

Section 1: Propositional Logic

Formal Definition: Proposition

"The **proposition expressed by a sentence** is the **set of possible cases [situations]** of which that sentence is true."
Zimmermann & Sternefeld (2013), p. 141.

Coin-flip example:

situation	flip1	flip2
1	heads	heads
2	tails	tails
3	heads	tails
4	tails	heads

Sentence	**Proposition**
S_1: only one flip landed heads up	$[\![S_1]\!] = \{3, 4\}$
S_2: all flips landed heads up	$[\![S_2]\!] = \{1\}$
S_3: flips landed at least once tails up	$[\![S_3]\!] = \{2, 3, 4\}$
etc.	etc.

Propositional Formulas

"The propositional letters and the **composite expressions** which are formed from them by means of connectives are grouped together as *sentences* or **formulas**. We designate these by means of the letters ϕ and ψ, etc. For these **metavariables**, unlike the variables p, q, and r, there is no convention that different letters must designate different formulas."

Gamut, L.T.F (1991). Volume 1, p. 29.

Examples:

$\phi \equiv \mathrm{p}, \mathrm{q}, \mathrm{r}$, etc.

$\phi \equiv \neg\mathrm{p}, \neg\mathrm{q}, \neg\mathrm{r}$, etc.

$\phi \equiv \mathrm{p} \wedge \mathrm{q}, \mathrm{p} \vee \mathrm{q}$, etc.

$\phi \equiv \neg(\neg\mathrm{p}_1 \vee \mathrm{q}_5) \rightarrow \mathrm{q}$, etc.

The Vocabulary

We can now define a **language L for propositional logic**. The "vocabulary" A of L consits of the propositional letters (e.g. p, q, r, etc.), the operators (e.g. $\neg$, $\wedge$, $\vee$, $\rightarrow$, etc.), as well as the round brackets '(' and ')'. The latter are important to group certain letters and operators together. We thus have:

$$A = \{p, q, r, ..., \neg, \wedge, \vee, \rightarrow, ..., (,)\} \quad (1)$$

The Syntax: Recursive Definition

Reminiscent of formal grammars of natural languages (see last years lecture on Phrase Structure Grammar), we now also need to define **syntactic rules** which allow for the symbols of the vocabulary to be combined yielding **well-formed expressions**. These rules are:

(i) Propositional letters in the vocabulary of L are formulas in L.

(ii) If ϕ is a formula in L, then $\neg\phi$ is too.

(iii) If ϕ and ψ are formulas in L, then $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$ are too.[1]

(iv) Only that which can be generated by the clauses (i)-(iii) in a finite number of steps is a formula in L.

Gamut, L.T.F (1991). Volume 1, p. 35.

[1] We could also add the *exclusive or* here as a connective.

Examples of Valid and Invalid Formulas

Formula	Rule Applied
p ✓	(i)
¬¬¬q ✓	(i) and (ii)
$((\neg p \wedge q) \vee r)$ ✓	(i), (ii), and (iii)
$((\neg(p \vee q) \rightarrow \neg\neg\neg q) \leftrightarrow r)$ ✓	(i), (ii), and (iii)
pq x	–
$\neg(\neg\neg p)$ x	–
$\wedge p \neg q$ x	–
$\neg((p \wedge q \rightarrow r))$ x	–

The Semantics of Propositional Logic

"The valuations we have spoken of [i.e. truth valuations of formulas] can now, in the terms just introduced [i.e. functions], be described as (unary)[2] **functions mapping formulas onto truth values**. But not every function with formulas as its domain and truth values as its range will do. A valuation must agree with the **interpretations of the connectives** which are given in their truth tables."

Gamut, L.T.F (1991). Volume 1, p. 35.

[2]An *unary* function is a function with a single argument, e.g. f(x). A *binary* function could be f(x,y), a *ternary* function f(x,y,z), etc.

Valuation Function

The valuation function V for each logical operator and logical formulas ϕ and ψ are then given as:

(i) Negation: $V(\neg\phi) = 1$ iff $V(\phi) = 0$,

(ii) Logical "and": $V(\phi \wedge \psi) = 1$ iff $V(\phi) = 1$ and $V(\psi) = 1$,

(iii) Inclusive "or": $V(\phi \vee \psi) = 1$ iff $V(\phi) = 1$ or $V(\psi) = 1$,

(iv) Material implication: $V(\phi \to \psi) = 0$ iff $V(\phi) = 1$ and $V(\psi) = 0$,

(v) Material equivalence: $V(\phi \leftrightarrow \psi) = 1$ iff $V(\phi) = V(\psi)$.

Gamut (1991). Volume I, p. 44.

Section 2: Predicate Logic

Propositional Logic vs. Predicate Logic

Commonalities:

- Usage of the same connectives and negation.

Differences:

- The introduction of **constants and variables** representing individuals and predicates to capture the main structural building blocks of sentences.
- The introduction of **quantifiers** to allow for quantified statements.

The Vocabulary

Similar as for propositional logic, we can define a **language** *L* **for predicate logic**. In this case, the "vocabulary" of *L* consits of

- a (potentially infinite) supply of **constant symbols** (e.g. a, b, c, etc.),
- a (potentially infinite) supply of **variable symbols** representing the constants (e.g. x, y, z, etc.),
- a (potentially infinite) supply of **predicate symbols** (e.g. A, B, C, etc.),
- the **connectives** (e.g. $\neg$, $\wedge$, $\vee$, $\rightarrow$, etc.),
- the **quantifiers** $\forall$ and $\exists$,
- as well as the round brackets '(' and ')'.
- (The equal sign '='.)

Translation Key

In order to translate a set of natural language sentences into predicate logic expressions unambiguously, we need a **translation key** listing the **predicates** and **constant symbols**.

Gamut, L.T.F (1991). Volume 1, p. 68.

English sentences:

(1) John is bigger than Peter or Peter is bigger than John.

(2) Alkibiades does not admire himself.

(3) If Socrates is a man, then he is mortal.

(4) Ammerbuch lies between Tübingen and Herrenberg.

(5) Socrates is a mortal man.

Translation key:

a_1: Alcibiades
a_2: Ammerbuch
j: John
p: Peter
s: Socrates
t: Tübingen
h: Herrenberg

Axy: x admires y
B_1xy: x is bigger than y
B_2xyz: x lies between y and z
M_1x: x is a man
M_2x: x is mortal

Translation Examples

We can then translate the natural language sentences into predicate logic by further identifying the logical operators, i.e. connectives and negation.

Gamut, L.T.F (1991). Volume 1, p. 68.

English sentences:

(1) John is bigger than Peter **or** John is bigger than Socrates.

(2) Alcibiades does **not** admire himself.

(3) **If** Socrates is a man, **then** he is mortal.

(4) Ammerbuch lies between Tübingen and Herrenberg.

(5) Socrates is a mortal man.

Translations:

(1) $B_1jp \vee B_1js$

(2) $\neg Aa_1a_1$

(3) $M_1s \rightarrow M_2s$

(4) B_2a_2th

(5) $M_1s \wedge M_2s$

The Syntax: Recursive Definition

Given the vocabulary of *L* we define the following clauses to create formulas of *L*.

(i) If A is an n-ary predicate letter in the vocabulary of *L*, and each of $t_1, \ldots, t_n$ is a constant or a variable in the vocabulary of *L*, then $At_1, \ldots, t_n$ is a formula in *L*.

(ii) If ϕ is a formula in *L*, then $\neg\phi$ is too.

(iii) If ϕ and ψ are formulas in *L*, then $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$ are too.

(iv) If ϕ is a formula in *L* and *x* is a variable, then $\forall x\phi$ and $\exists x\phi$ are formulas in L.

(v) Only that which can be generated by the clauses (i)-(iv) in a finite number of steps is a formula in *L*.

Gamut, L.T.F (1991). Volume 1, p. 75.

Examples of Valid and Invalid Formulas

Formula	Rule Applied
Aa ✓	(i)
Ax ✓	(i)
Aab ✓	(i)
Axy ✓	(i)
$\neg$Axy ✓	(i) and (ii)
Aa$\rightarrow$Axy ✓	(i) and (iii)
$\forall$x(Aa$\rightarrow$Axy) ✓	(i),(iii), and (iv)
$\forall$xAa$\rightarrow$Axy ✓	(i),(iii), and (iv)
a x	–
A x	–
$\forall$ x	–
$\forall$(Axy) x	–

Definition: The valuation function $\mathbf{V}_M$

"If **M** is a model for L whose interpretation function I is a function of the constants in L onto the domain D, then $\mathbf{V}_M$, the valuation V based on M, is defined as follows:"

(i) If $Aa_1, \ldots, a_n$ is an atomic sentence in L, then $\mathbf{V}_M(Aa_1, \ldots, a_n) = 1$ if and only if $\langle I(a_1), \ldots, I(a_n)\rangle \in I(A)$.

(ii) $V_M(\neg\phi) = 1$ iff $V_M(\phi) = 0$.

(iii) $V_M(\phi \wedge \psi) = 1$ iff $V_M(\phi) = 1$ and $V_M(\psi) = 1$.

(iv) $V_M(\phi \vee \psi) = 1$ iff $V_M(\phi) = 1$ or $V_M(\psi) = 1$.

(v) $V_M(\phi \rightarrow \psi) = 0$ iff $V_M(\phi) = 1$ and $V_M(\psi) = 0$.

(vi) $V_M(\phi \leftrightarrow \psi) = 1$ iff $V_M(\phi) = V_M(\psi)$.

Gamut, L.T.F (1991). Volume 1, p. 91.

Definition: The valuation function $\mathbf{V}_M$

(vii) $V_M(\forall x\phi) = 1$ iff $V_M([c/x]\phi) = 1$ for all constants c in L.

(viiii) $V_M(\exists x\phi) = 1$ iff $V_M([c/x]\phi) = 1$ for at least one constant c in L.

If $V_M(\phi) = 1$, then ϕ is said to be true in model **M**.

Gamut, L.T.F (1991). Volume 1, p. 91.

Note: The notation [c/x] means “replacing x by c”. Note that this valuation works only for **sentences** of predicate logic as defined above. That is, it works for formulas that consist of atomic sentences and/or formulas with variables that are bound. For *formulas with free variables*, it does not work.

Valuation Example

Given a Model of the world **M**, consisting of D and I, and some formula ϕ which adheres to predicate logic syntax (and which consists of atomic sentences and or quantifications with bound variables), we can then evaluate the truth of ϕ as follows.

Model **M**

$D = \{e_1, e_2, e_3\}$
$I = \{\langle j, e_1\rangle, \langle p, e_2\rangle, \langle m, e_3\rangle, \langle S, \{\langle I(j), I(m)\rangle, \langle I(p), I(m)\rangle\}\rangle\}$
$I(S) = \{\langle I(j), I(m)\rangle, \langle I(p), I(m)\rangle\}$
Translation key: j: John; p: Peter; m: morning star; Sxy: x sees y.

Valuation

"John sees the morning star": V_M(Sjm) = 1 (according to (i))
"Everybody sees the morning star": V_M(∀xSxm) = 0 (according to (vii))[3]

[3] This valuation gives 0 since the morning star (m) is a constant c in L, but it does not see itself, i.e. $\langle I(m), I(m)\rangle \notin S$.

Section 3: Second-Order Logic

First-Order Logic vs. Second-Order Logic

Commonalities:

- Usage of the same **logical operators** (connectives, negation, quantifiers).
- Generally similar syntax and valuation of expressions.

Differences:

- Introducing **first-order predicate variables**, and **second-order predicates**.

Beyond Predicate Logic

We have seen that predicate logic is an extension of propositional logic, by introducing predicates and quantifiers. **Predicate logic** might itself be superseded by another logical system, called **second-order logic**.

Gamut, L.T.F (1991). Volume 1, p. 168.

Take the following English sentences:

(1) Mars is red.
(2) Red is a color.
(3) Mars has a color.
(4) John has at least one thing in common with Peter.

How can we translate these into logical expressions?

First-Order and Second-Order Logic

A **second-order logic** language L' is then an extension to a standard predicate logic language L by adding second-order predicates to L. The original language L is then sometimes referred to as **first-order logic** language.

Further Examples:

(5) $\exists X(\mathcal{C}X \wedge Xm)$ (English sentence: “Mars has a color.”)

(6) $\exists X(Xj \wedge Xp)$ (English sentence: “John has at least one thing in common with Peter.”)

(7) $\exists \mathcal{X}(\mathcal{X}R \wedge \mathcal{X}G)$ (English sentence: “Red has something (a property) in common with green.”)

Vocabulary (special to Second-Order Logic)

The vocabulary extensions to fit **second-order logic requirements** are:

- A (potentially infinite) supply of **first-order predicate variables** (e.g. X, Y, Z, etc.), which are necessary to quantify over first-order predicates,
- a (potentially infinite) supply of **second-order predicate constants** (e.g. $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, etc.).

If we wanted to take it even at a higher-order level we could also have:

- a (potentially infinite) supply of **second-order predicate variables** (e.g. $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$, etc.) to stand in for second-order predicates.

The Syntax: Recursive Definition

Given the vocabulary of L we then define the following clauses to create formulas of *L*:

(i) If A is an n-ary **first-order** predicate letter/constant in *L*, and $t_1, \ldots, t_n$ are individual terms in *L*, then $At_1, \ldots, t_n$ is an (atomic) formula in *L*;

(ii) If X is a [**first-order**] predicate variable and t is an individual term in *L*, then Xt is an atomic formula in *L*;

(iii) If $\mathcal{A}$ is an n-ary **second-order** predicate letter/constant in *L*, and $T_1, \ldots, T_n$ are **first-order unary** predicate constants, or predicate variables, in *L*, then $\mathcal{A}T_1, \ldots, T_n$ is an (atomic) formula in *L*;

(iv) If ϕ is a formula in *L*, then $\neg\phi$ is too;

(v) If ϕ and ψ are formulas in *L*, then $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$ are too.

The Syntax: Recursive Definition

Given the vocabulary of L we then define the following clauses to create formulas of *L*:

(vi) If *x* is an individual variable ϕ is a formula in *L*, then $\forall x\phi$ and $\exists x\phi$ are also formulas in L;

(vii) If *X* is a [first-order] predicate variable, and ϕ is a formula in *L*, then $\forall X\phi$ and $\exists X\phi$ are also formulas in L;

(viii) Only that which can be generated by the clauses (i)-(vii) in a finite number of steps is a formula in *L*.

Gamut, L.T.F (1991). Volume 1, p. 170.

Note: In the above clauses (i) and (ii), the word “term” is used, which has not been defined by us before. In the context here, suffices to say that it includes both constants and variables (of constants), i.e. a, b, c, etc. and x, y, z, etc.

Examples of Valid and Invalid Formulas

Formula	Rule Applied
Aa ✓	(i)
Ax ✓	(i)
Axy ✓	(i)
Xa ✓	(ii)
Xx ✓	(ii)
$\mathcal{A}$A ✓	(iii)
Xa$\rightarrow\neg$Xb ✓	(ii), (iv) and (v)
$\forall$X$\forall$x(Xa$\rightarrow$Axy) ✓	(i),(ii), (v), (vi), and (vii)
x x	–
X x	–
Xab x	–
$\forall$(Xa) x	–

Section 4: Type Theory

Standard (First-Order) Logic vs. Typed Logic

Commonalities:

- Usage of the same **logical operators** (connectives, negation, quantifiers).

Differences:

- Introduction of a **potentially infinite number of types** defined for logical constants and variables which we can quantify over. Note that this makes typed logic a **higher-order logic**.

Application to Natural Language

We can apply the theory of types to a logical language L by first defining the **two most basic types**, of which all other types are **composed**. These are the type e for **entities**, i.e. individual constants (e.g. John, Jumbo), and the type t for **sentences**, where t stands for *truth*, since truth values can only be assigned to sentences.

In the following we will expose how the **syntax** of a type-theoretic logical language L is defined.

Definition: The Syntax of Types

For the set of types $\mathbb{T}$ we define that:

(i) $e, t \in \mathbb{T}$,

(ii) if $a, b \in \mathbb{T}$, then $\langle a, b\rangle \in \mathbb{T}$,

(iii) nothing is an element of $\mathbb{T}$ except on the basis of clauses (i) and (ii).

Gamut (1991), Volume 2, p. 79.

Note: *a* and *b* above are variables which stand in for all kinds of types. This means we can create an infinite number of types by recursively applying clause (ii). For example:

Applying (ii) to $a = e$ and $b = t$ yields $\langle e, t\rangle$
Applying (ii) to $a = \langle e, t\rangle$ and $b = t$ yields $\langle\langle e, t\rangle, t\rangle$
Applying (ii) to $a = e$ and $b = \langle\langle e, t\rangle, t\rangle$ yields $\langle e, \langle\langle e, t\rangle, t\rangle\rangle$
etc.

Examples of Valid and Invalid Types

e ✓
t ✓
$\langle e, t \rangle$ ✓
$\langle t, e \rangle$ ✓
$\langle t, \langle t, e \rangle \rangle$ ✓
$\langle \langle t, \langle t, e \rangle \rangle, t \rangle$ ✓

et x
e, t x
$\langle e, e, t \rangle$ x
$\langle e, \langle e, t \rangle$ x

Note: The usage of left and right ankled brackets as defined by clause (ii) results in a **strict binarization** of the internal structure of types, i.e. at each level of embedding we always have an **ordered pair** of more basic types.

Definition: Functional Application

How do we derive one type of expression from another?

“[...] if α is an expression of type $\langle a, b\rangle$ and β is an expression of type a, then $\alpha(\beta)$ is of type b.”

Gamut (1991), Volume 2, p. 79.

Examples

If $\alpha = \langle e, t\rangle$ and $\beta = e$ then $\alpha(\beta) = t$.
If $\alpha = \langle\langle e, t\rangle, \langle e, t\rangle\rangle$ and $\beta = \langle e, t\rangle$ then $\alpha(\beta) = \langle e, t\rangle$.
If $\alpha = \langle t, \langle t, e\rangle\rangle$ and $\beta = t$ then $\alpha(\beta) = \langle t, e\rangle$.

However,

If $\alpha = \langle t, \langle t, e\rangle\rangle$ and $\beta = \langle t, e\rangle$ then $\alpha(\beta)$ is **not defined**.

Notation for Variables and Constants

As before, we will use the following notations to distinguish typographically between different variables and constants at different orders:

- Constants for entities: a, b, c, etc.
- Variables over entities: x, y, z, etc.
- First-order predicate constants: A, B, C, etc.
- Variables over first-order predicates: X, Y, Z, etc.
- Second-order predicate constants: $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, etc.
- (Second-order predicate variables: $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$, etc.)[4]

[4]These are just added for completeness here. We generally don't go into orders higher than *two* in exercises and examples.

The Syntax: Recursive Definition

The clauses for the syntax of a type-theoretic language are then:

(i) If α is a variable or a constant of type a in L [i.e. v_a or c_a], then α is an expression of type a in L.

(ii) If α is an expression of type $\langle a, b \rangle$ in L, and β is an expression of type a in L, then $(\alpha(\beta))$ is an expression of type b in L.

(iii) If ϕ and ψ are expressions of type t in L (i.e. formulas in L), then so are $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$.

(iv) If ϕ is an expression of type t in L and v is a variable (of arbitrary type a), then $\forall v\phi$ and $\exists v\phi$ are expression of type t in L.

(v) If α and β are expressions in L which belong to the same (arbitrary) type, then $(\alpha = \beta)$ is an expression of type t in L.

(vi) Every expression L is to be constructed by means of (i)-(v) in a finite number of steps.

Gamut (1991), Volume 2, p. 81-82.

Examples of Valid and Invalid Expressions

Definition of Types

Assume j is of type e (i.e. representing an entity), x is of type e, A is of type $\langle e, t \rangle$ (i.e. a first order one-place predicate), B is of type $\langle e, \langle e, t \rangle \rangle$ (i.e. a first-order two-place predicate), and $\mathcal{C}$ is of type $\langle \langle e, t \rangle, t \rangle$ (i.e. a second-order one-place predicate).

Expressions	**Clause Applied**
j ✓	(i)
A ✓	(i)
A(j) ✓	(i) and (ii)
(B(j))(x) ✓ alternative notation: B(j)(x)	(i) and (ii)
$\mathcal{C}$(B(j)) ✓	(i) and (ii)
A(j) $\wedge$ $\mathcal{C}$(A) ✓	(i) and (ii)
∀xA(x) ✓	(i), (ii), and (iii)
	(i), (ii), and (iv)
Aj x	
B(A) x	–
∀x$\mathcal{C}$(x) x	–
	–

Section 1: Propositional Logic
Section 2: Predicate Logic
Section 3: Second-Order Logic
Section 4: Type Theory
Section 5: λ-calculus
Summary
References

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Philosophy
General Linguistics

Section 5: λ-calculus

The Syntax: Adding the λ-clause

We simply add another clause to the **type-theoretic language** syntax:

(vii) If α is an expression of type a in L, and v is a variable of type b, then λv(α) is an expression of type $\langle b, a \rangle$ in L.[5]

Gamut (1991), Volume 2, p. 104.

[5] I added the brackets around α here, since at least in some cases these are necessary to disambiguate.

Examples of λ-Abstractions

Assume a, b and x, y are of type *e*; A is of type $\langle e, t\rangle$; B is of type $\langle e, \langle e, t\rangle\rangle$; and X is of type $\langle e, t\rangle$.

Expressions	Types	λ-Abstraction	Types
x ✓	e	λx(x)	$\langle e, e\rangle$
A(x)✓	t	λx(A(x))	$\langle e, t\rangle$
B(y)(x) ✓	t	λx(B(y)(x)) or λy(B(y)(x))	$\langle e, t\rangle$
B(a)(x) ✓	t	λx(B(a)(x))	$\langle e, t\rangle$
$\forall$xB(x)(y)✓	t	λy($\forall$xB(x)(y))	$\langle e, t\rangle$
X(a) ✓	t	λX(X(a))	$\langle\langle e, t\rangle, t\rangle$
X(a) $\wedge$ X(b)✓	t	λX(X(a) $\wedge$ X(b))	$\langle\langle e, t\rangle, t\rangle$

Note: In our practical usage of the type-theoretic language, variables are mostly defined to have type *e* (i.e. x, y, z, etc.). In some cases, they might be of type $\langle e, t\rangle$, namely, if they refer to predicate variables (X, Y, Z, etc.). Hence, λ-abstraction essentially amounts to **adding an *e* or $\langle e, t\rangle$ as a "prefix"** to the type of the expression that is abstracted over.

λ-Conversion (aka β-Reduction)

Informally speaking, **λ-conversion**[6] is the process whereby we reduce the λ-statement by removing the λ-operator (and the variable directly following it) and pluging an expression (in the simplest case a constant c, or a predicate constant C) into **every occurrence of the variable which is bound by the λ-operator**.

Typed expression	**λ-Abstraction (over x or X)**	**λ-Conversion (with c or C over x or X)**
$S(x)$	$\lambda x(S(x))$	$\lambda x(S(x))(c) = S(c)$
$S(x) \wedge D(x)$	$\lambda x(S(x) \wedge D(x))$	$\lambda x(S(x) \wedge D(x))(c) = S(c) \wedge D(c)$
$X(a) \wedge X(b)$	$\lambda X(X(a) \wedge X(b))$	$\lambda X(X(a) \wedge X(b))(C) = C(a) \wedge C(b)$

[6] The term λ-conversion is not to be confused with α-conversion. The latter refers to replacing one variable for another.

Why is λ-calculus needed?

If our aim is to model not only full sentences and formulas representing predicates, but also parts of sentences, and even individual words, by using in a unified account, then λ-abstraction and λ-conversion are possible solutions. Thus, λ-calculus allows us to capture the **compositionality of language**.

English sentence	**Typed expression**
John smokes and drinks.	$\lambda x(S(x) \wedge D(x))(j) = S(j) \wedge D(j)$
John smokes	$\lambda x(S(x))(j) = S(j)$
smokes	$\lambda x(S(x))$
drinks	$\lambda x(D(x))$
John	$\lambda X(X(j))$
smokes and drinks	$\lambda x(S(x) \wedge D(x))$

Summary

Translation Summary

Natural Language	PL	FOL	SOL	TL
John smokes.	p	Sj	Sj	$S(j)$
John smokes and drinks.	$p \wedge q$	$Sj \wedge Dj$	$Sj \wedge Dj$	$S(j) \wedge D(j)$
Jumbo likes Bambi.	r	Ljb	Ljb	$L(b)(j)$
Every man walks.	p_1	$\forall x(Mx \rightarrow Wx)$	$\forall x(Mx \rightarrow Wx)$	$\forall x(M(x) \rightarrow W(x))$
Red is a color.	q_1	Cr	$\mathcal{C}R$	$\mathcal{C}(R)$
smokes and drinks	_	_	_	$\lambda x(S(x) \wedge D(x))$
every man	_	_	_	$\lambda X(\forall x(M(x) \rightarrow X(x)))$
every	_	_	_	$\lambda Y(\lambda X(\forall x(Y(x) \rightarrow X(x))))$
is	_	_	_	$\lambda X(\lambda x(X(x)))$

PL: Propositional Logic
FOL: First-Order Predicate Logic
SOL: Second-Order Predicate Logic
TL: Typed Logic (Higher-Order) with λ-calculus

References

References

Gamut, L.T.F (1991). *Logic, Language, and Meaning. Volume 1: Introduction to Logic*. Chicago: University of Chicago Press.

Gamut, L.T.F (1991). *Logic, Language, and Meaning. Volume 2: Intensional Logic and Logical Grammar*. Chicago: University of Chicago Press.

Thank You.

Contact:

Faculty of Philosophy
General Linguistics
Dr. Christian Bentz
SFS Wihlemstraße 19-23, Room 1.24
chris@christianbentz.de
Office hours:
During term: Wednesdays 10-11am
Out of term: arrange via e-mail