EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Philosophy
General Linguistics

Semantics & Pragmatics SoSe 2021

Lecture 7: Formal Semantics IV (Type Theory)

20/05/2021, Christian Bentz

Overview

© 2012 Universität Tübingen

Section 1: Recap of Lecture 6

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 **Predicate logic** might itself be superseded by another logical system, called **second-order logic**.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?

© 2012 Universität Tübingen

Second-Order Predicates

To circumvent this discrepancy, we can construe the predicate *x is a color* not as a property, but as a **property of properties**. $\mathcal{C}$ then represents a so-called **second-order property**, i.e. a **second-order predicate** over the first-order predicate *x is red*.

Instead of

(5) Cr (Cx: x is a color, r: red),

we then get

(6) $\mathcal{C}$R ($\mathcal{C}$X: X is a predicate with the property of being a color, Rx: x is red)

Note: We introduce **two** new sets of symbols here compared to standard predicate logic, a) the set of second-order predicates

© 2012 Universität Tübingen

Second-Order Logic

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

Further Examples:

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

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

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

© 2012 Universität Tübingen

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.

© 2012 Universität Tübingen

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.

© 2012 Universität Tübingen

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.

© 2012 Universität Tübingen

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	–

© 2012 Universität Tübingen

Summary

- **Second-order predicate logic** goes beyond first-order predicate logic by, firstly, introducing **predicate variables**, which allow to quantify over first-order predicates, and secondly, by introducing **second order predicates**, which are to be seen as properties of properties, i.e. predicates over predicates.
- These changes lead to **adjustments in the formal definitions** of the syntax and semantics of the logical language L.
- These adjustments enable the translation of a wider array of natural language sentences, although there are still natural language phenomena not captured appropriately.

© 2012 Universität Tübingen

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Philosophy
General Linguistics

Section 2: Beyond Standard Logic

Beyond Standard Logic

Throughout the last lectures we have seen that propositional logic is superseeded by first-order predicate logic, while this is in turn superseeded by second-order predicate logic. Propositional logic and predicate logic are typically grouped together as **Standard Logic**.

As we move from propositional to predicate logic, and then beyond standard logic, we increasingly tease apart sentences into their **component parts**.

© 2012 Universität Tübingen

Example

All animals that live in the jungle have a color.

Propositional logic:
p

First-order predicate logic:
$\forall x((Ax \wedge Jx) \rightarrow Cx)$
Translation key: Ax: x is an animal; Jx: x lives in the jungle; Cx: x has a color.

Second-order predicate logic:
$\forall x(\exists X((\mathcal{A}X \wedge Xx) \wedge Jx) \rightarrow \exists Y(Yx \wedge \mathcal{C}Y))$
Translation key: $\mathcal{A}$X: x is a property (type of animal) which has the property of being an animal; Jx: x lives in the jungle; $\mathcal{C}$X: X is a property (a particular color) which has the property of being a color.

© 2012 Universität Tübingen

The Problem of Semantic Compositionality

However, in both first- and second-order logic, there are still no tools to get to grips with frequent **compositional structures** in natural language:

- **adjective-noun** combinations
- **adverb-verb** combinations
- **article-noun** combinations
- **prepositions-NP** combinations
- etc.

© 2012 Universität Tübingen

Example: Adjectives and Nouns

Adjectives and nouns frequently combine to yield compound expressions in which the compound meaning is a combination of the individual meanings (e.g. *pink elephant*).

(10) *Jumbo is a pink elephant.*
First-Order Predicate Logic: Ej $\wedge$ Pj

(11) *Jumbo is a small elephant.*
First-Order Predicate Logic: Ej $\wedge$ Sj

Note: In the first case, the predicate logic expression might reflect the English sentence correctly. In the second example, namely, when a *relative adjective* (e.g. small, big, fast, slow, etc.) is used, the English sentence is missrepresented, since Jumbo might be small for an elephant, but not small in general. The adjective here modifies the noun and results in a new predicate *small elephant*.

Gamut (1991), Volume 2, p. 77.

© 2012 Universität Tübingen

Example: Adverbs and Verbs

Adverbs and verbs likewise combine to yield compound expressions. Again, the available predicate logic translation falls short of the actual meaning of *walking quickly*. The adverb modifies the verb and creates a new predicate.

(12) *Jumbo is walking quickly.*

(13) *Jumbo is walking and Jumbo is quick.*
Predicate logic: Wj $\wedge$ Qj

Gamut (1991), Volume 2, p. 77.

© 2012 Universität Tübingen

Types of Expressions

There is a long (potentially infinite) list of **types of expressions** which we might want to represent in our logical language in order to capture the different combinatorial possibilities we find in natural languages.

Kind of expression	**Examples**
Individual expression	*John, Jumbo*
One-place first-order predicate	*walks, red, loves Mary*
Two-place first-order predicate	*loves, lies between Amsterdam and*
Three-place first-order predicate	*lies between (and)*
Sentence	*John walks, John loves Mary*
Sentential modifier	*not*
Function	*the father of*
Predicate modifier	*quickly, beautifully, fast*
One-place second-order predicate	*is a color*
Two-place second-order predicate	*is a brighter color than*
etc.	etc.

© 2012 Universität Tübingen

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

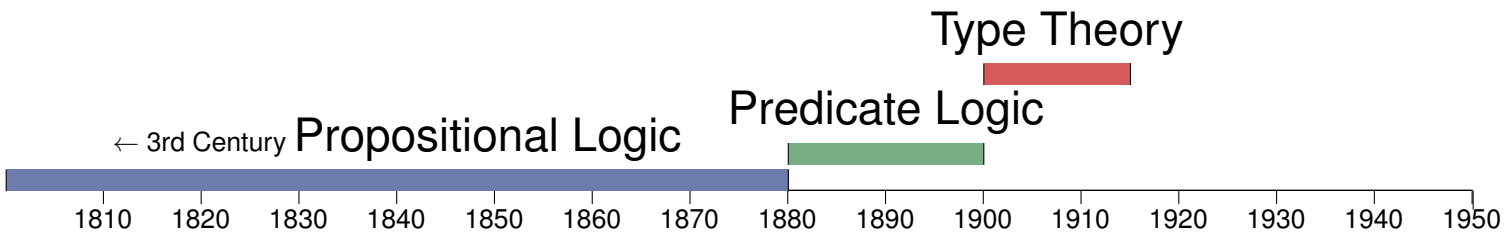
Faculty of Philosophy
General Linguistics

Section 3: The Theory of Types

The Theory of Types

How can we represent this **potentially infinite number of expressions** while preserving their *internal structure* and *combinatorial relationships*? – A logical system developed to fit this requirement is the so-called **theory of types** which was developed by Bertrand Russell as a remedy for paradoxes encountered in set theory.

Gamut (1991), Volume 2, p. 78.



© 2012 Universität Tübingen

Historical Note: Russell's Paradox

- There are sets which contain themselves, e.g. $\mathbb{U} = \{x|x = x\}$, which is called the *universal set*, and for which it holds that $\mathbb{U} \in \mathbb{U}$. It contains everything that exists, since everything is equal to itself.
- However, most familiar sets do not contain themselves, e.g. the set of natural numbers $\mathbb{N} = \{1, 2, 3, ...\}$, for which $\mathbb{N} \notin \mathbb{N}$, since a set is not a natural number.
- We thus might want to define the set of entities which are not members of themselves $\mathbb{R} = \{x|x \notin x\}$. A paradoxical question arises: is $\mathbb{R}$ a member of itself or not?
- Consider the so-called *Barber Paradox* as a more intuitive illustration: *Assume a barber shaves all those who do not shave themselves. – Does the barber shave himself?*

Gamut (1991), Volume 2, p. 78.

© 2012 Universität Tübingen

Historical Note: Russell's Solution

Each entity is assigned a **type** which represents the "level" of the logical language where the logical expression is located at. For instance, a natural number n is an individual entity. The set of natural numbers $\mathbb{N}$, on the other hand, is of the type "set of individual entities".

The membership relation is then defined to only apply to logical expressions which are **exactly one level apart**. Hence, $n \in \mathbb{N}$ is a valid expression in the logical language of types, while $\mathbb{N} \in \mathbb{N}$ is not.

© 2012 Universität Tübingen

Type Theory

"Linguistic expressions are classified into their **semantic types** according to the kind of denotation they have. The two most basic denotation types are **type e**, the type of **entities**, and **type t**, the type of **truth values**."

Kearns (2011). Semantics, p. 57.

Type of expression	**Type of extension**	**Semantic type**	**Example**
proper name	individual (entity)	**e**	$[\![\text{Paul}]\!]_s$=Paul McCartney
...	...	...	...
sentence	truth value	**t**	$[\![\text{Paul is happy}]\!]_s \in \{0, 1\}$

© 2012 Universität Tübingen

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.

© 2012 Universität Tübingen

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.

© 2012 Universität Tübingen

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**.

© 2012 Universität Tübingen

Functional Application in Natural Language Semantics

"[...] a function binds arguments together into a statement. From this insight, Frege proposed that all **semantic composition** is **functional application**. Functional application is just the combination of a function with an argument."

Kearns (2011), p. 58.

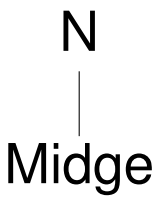
© 2012 Universität Tübingen

Semantic Types: Individuals

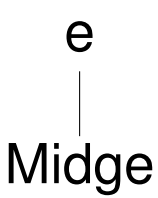
An **individual**, represented by a logical constant of the semantic **type** *e*, corresponds to, for instance, a proper noun (N) in natural language.

The following examples are based on Kearns (2011).

Syntactic Tree



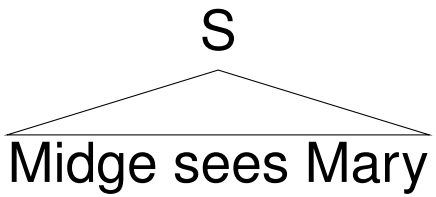
Type-Theoretic Tree

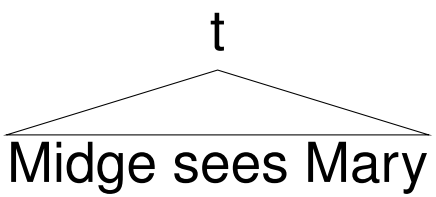


© 2012 Universität Tübingen

Semantic Types: Sentences

Logical formulae of the semantic **type** *t* correspond to sentences (S) in natural language.

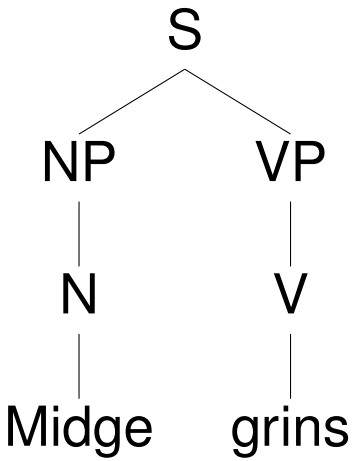


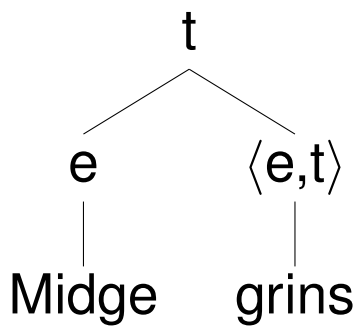


© 2012 Universität Tübingen

Semantic Types: One-Place Predicates

An **intransitive verb** requires **one argument** to be filled in order to form a full sentence, hence it is of the **type** ⟨**e,t**⟩. Remember that the argument is on the left side of the tuple (ordered pair), hence the component of type *entity* (e) is left.

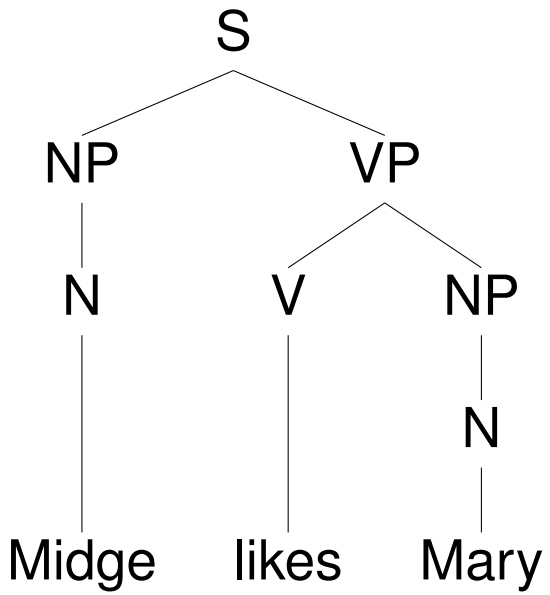


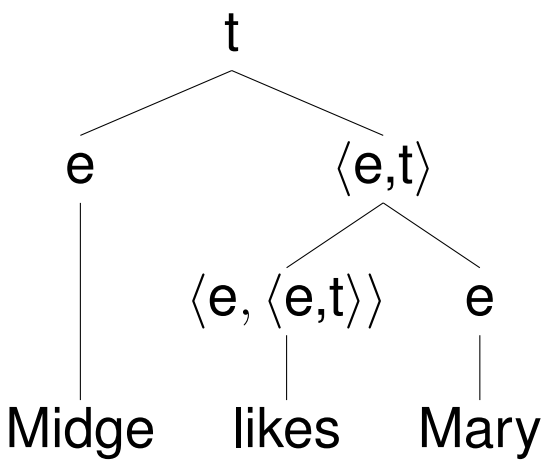


© 2012 Universität Tübingen

Semantic Types: Two-Place Predicates

A **transitive verb** requires **two arguments** to be filled in order to form a full sentence, hence it is of the **type** ⟨**e,**⟨**e,t**⟩⟩.

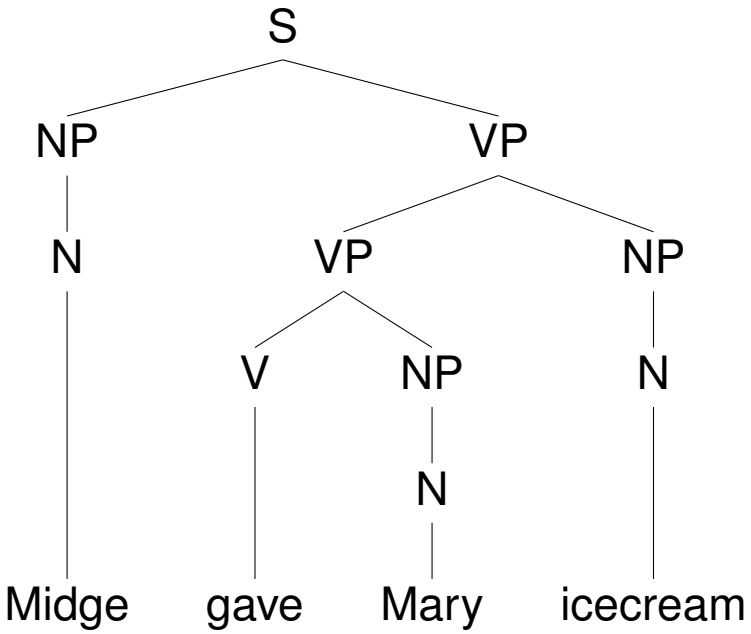


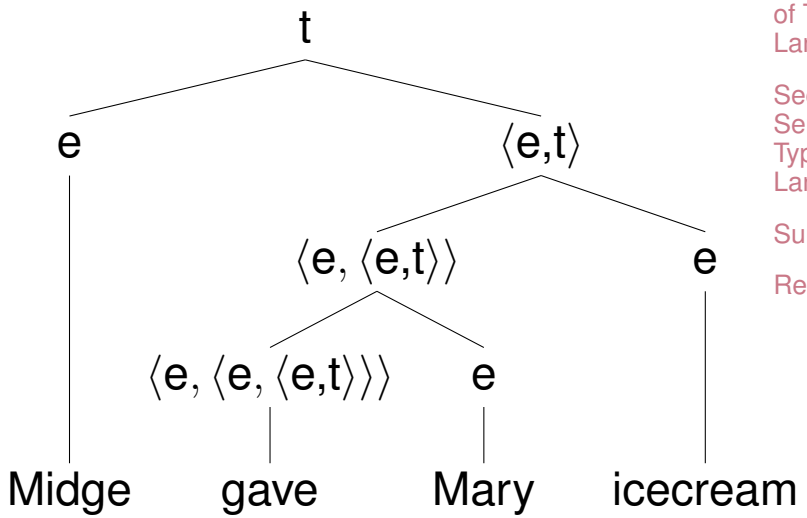


© 2012 Universität Tübingen

Semantic Types: Three-Place Predicates

A **ditransitive verb** requires **three arguments** to be filled in order to form a full sentence, hence it is of the **type** ⟨**e**,⟨**e**,⟨**e**,**t**⟩⟩⟩.

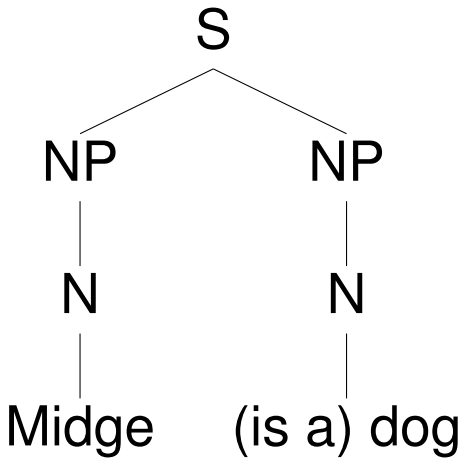


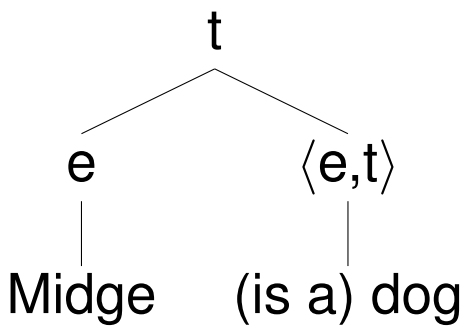


© 2012 Universität Tübingen

Semantic Types: Nouns

Common nouns are of type **type** ⟨**e,t**⟩. This might seem counterintuitive at first sight, but the idea here is that nouns are essentially like **one-place predicates**, in the sense that they require a concrete entity (e) to form a basic existential statement (with a copular) which can be true or false.

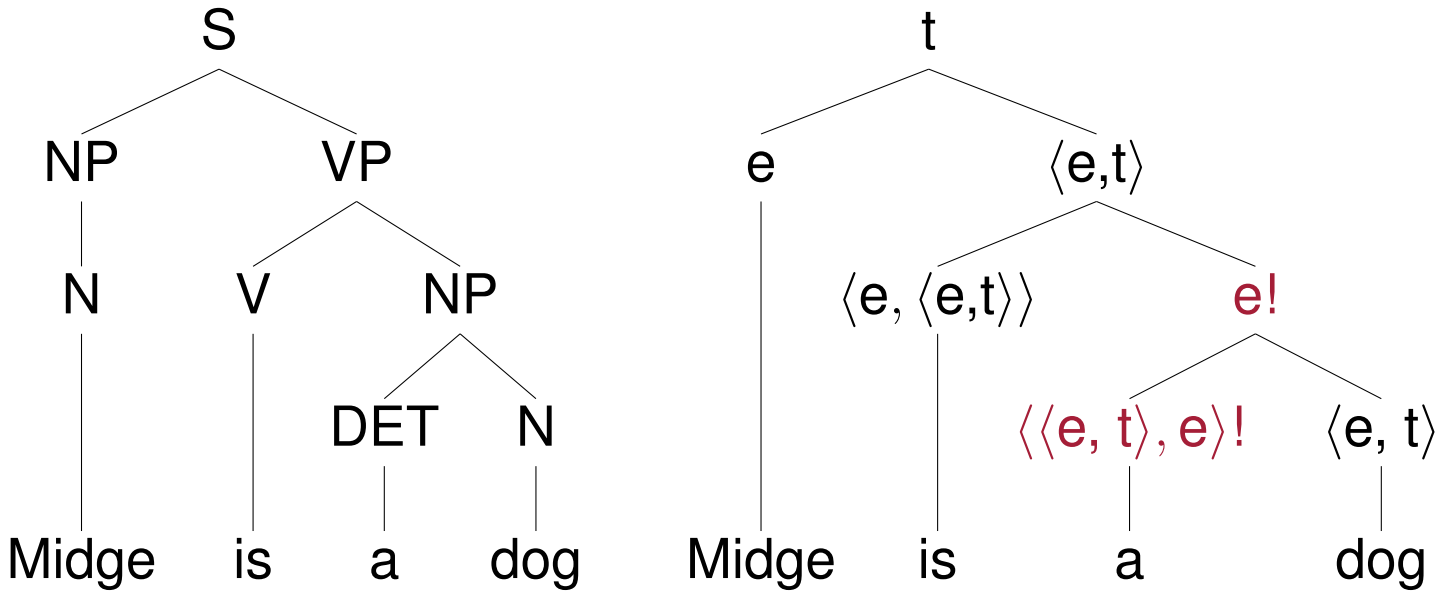




Note: This corresponds to the predicate logic formulation Dm, where the copula and the indefinite determiner are also dropped. As pointed out earlier in the syntax lectures, the copula is a problematic and controversial case, and the syntactic tree given here is unusual in that it does not have a verb.

© 2012 Universität Tübingen

Alternative Analysis?

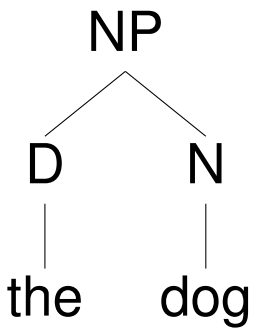


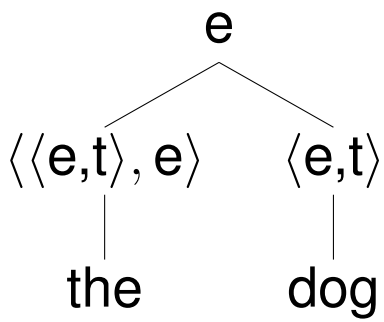
Note: This might seem like a valid alternative, but notice that the type of *a dog* has to be e now, meaning that it is an individual, rather than a set of individuals. So this would break with the fundamental definition that indeterminate expressions have sets as their extensions.

© 2012 Universität Tübingen

Semantic Types: NPs and Determiners

NPs are of **type e**, i.e. referring to a concrete entity. Note that it follows from this definition and the definition of common nouns above that **determiners** then have to be of **type** ⟨⟨**e,t**⟩, **e**⟩.

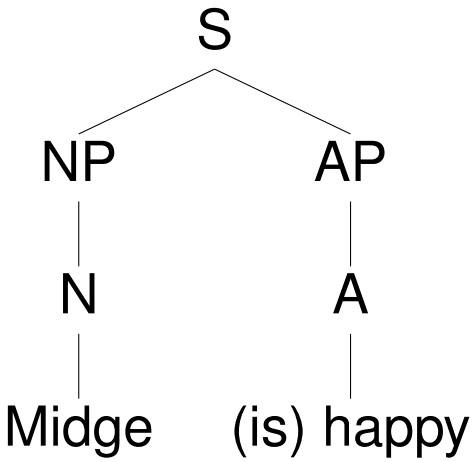


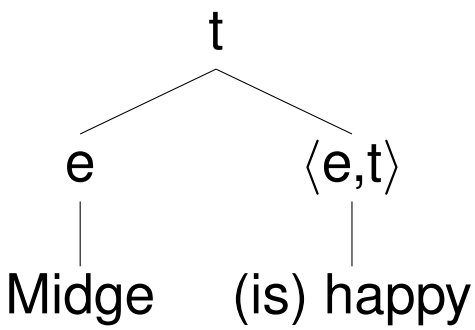


© 2012 Universität Tübingen

Semantic Types: Adjectives (Existential Statements)

Similar to common nouns, **adjectives** are considered to be of **type** ⟨**e,t**⟩. The same argument applies: they require a concrete entity (e) to form a basic existential statement (with a copular) which can be true or false.



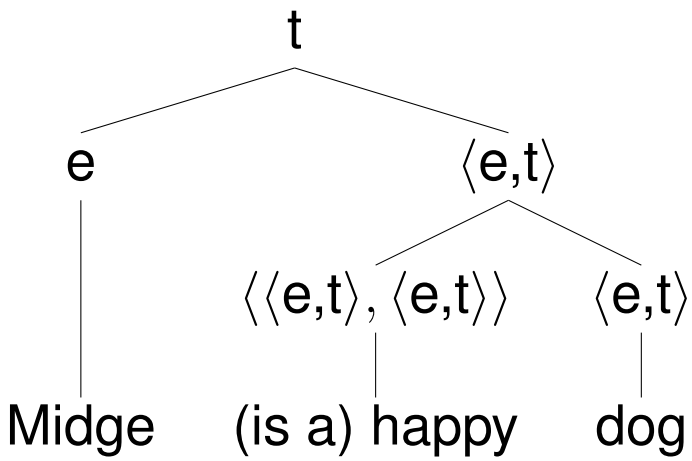


© 2012 Universität Tübingen

Semantic Types: Adjectives (Predicate Modifiers)

On the other hand, adjectives can also serve as **predicate modifiers**. In this case, they have to be defined as being of the type $\langle\langle$**e,t**$\rangle, \langle$**e,t**$\rangle\rangle$ (same as adverbs below).

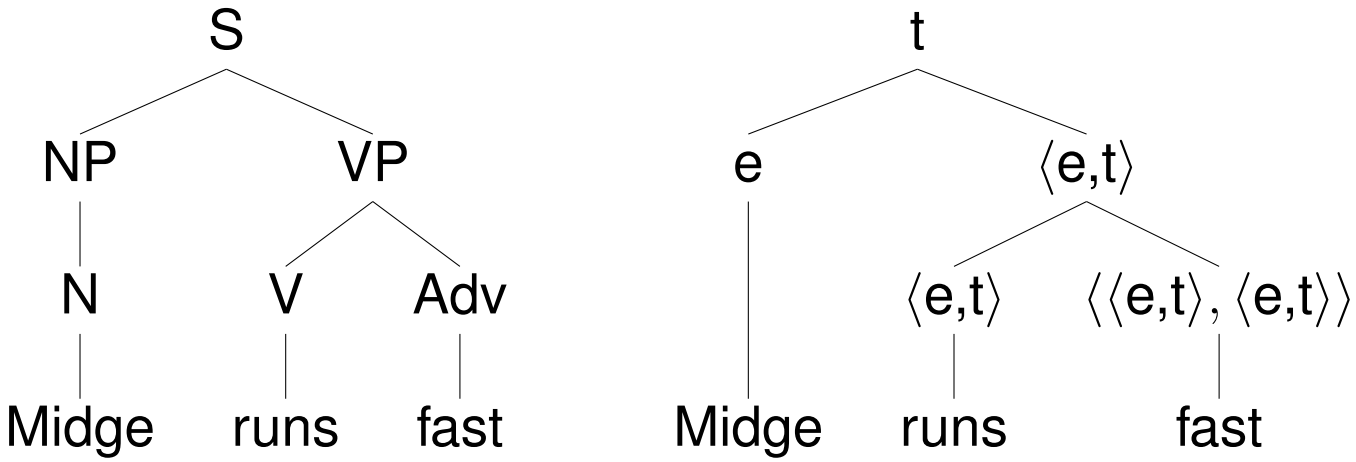
Gamut (1991), Volume 2, p. 77.



© 2012 Universität Tübingen

Semantic Types: Adverbs

Adverbs are considered **type** $\langle\langle$**e,t**$\rangle, \langle$**e,t**$\rangle\rangle$. Note that similar as for determiners, this is a logical consequence of the definition of other types, i.e. the definition of a one-place predicate modified by an adverb.



© 2012 Universität Tübingen

Summary: Types of Expressions

There is a long (potentially infinite) list of **types of expressions** which we might want to represent in our logical language in order to capture the different combinatorial possibilities we find in natural languages.

Type	**Kind of expression**	**Examples**
e	Individual expression	*John, Jumbo*
$\langle e, t\rangle$	One-place first-order predicate	*walks, red, loves Mary*
$\langle e, \langle e, t\rangle\rangle$	Two-place first-order predicate	*loves, sees*
$\langle e, \langle e, \langle e, t\rangle\rangle\rangle$	Three-place first-order predicate	*lies between (and)*
t	Sentence	*John walks, John loves Mary*
$\langle t, t\rangle$	Sentential modifier	*not*
$\langle e, e\rangle$	Function (entitiy to entity)	*the father of*
$\langle\langle e, t\rangle, \langle e, t\rangle\rangle$	Predicate modifier	*quickly, beautifully, fast*
$\langle\langle e, t\rangle, t\rangle$	One-place second-order predicate	*is a color*
$\langle\langle e, t\rangle, \langle\langle e, t\rangle, t\rangle\rangle$	Two-place second-order predicate	*is a brighter color than*
etc.	etc.	etc.

Based on Gamut (1991), Volume 2, p. 86.

© 2012 Universität Tübingen

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Philosophy
General Linguistics

Section 4: Syntax of Type-Theoretic Languages

Vocabulary (Shared)

The shared part of the vocabulary consists of:

- For every type a, an infinite set VAR_a of variables of type a;[1]
- the usual connectives $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$, $\neg$;
- the quantifiers $\forall$ and $\exists$;
- two brackets '(' and ')';
- the symbol for identity '='.[2]

Gamut (1991), Volume 2, p. 80.

[1] For example, for the type of individuals e we can define an infinite set of variable symbols x, y, z, etc. of this type (just as for predicate logic). Likewise, for the type of a one-place predicate $\langle e, t \rangle$ we can also define an infinite set of one-place first order predicate variables X, Y, Z, etc. (just as for predicate logic), and so on for every type.

[2] Optional as before, depending on whether we want to use expressions which reflect equations.

© 2012 Universität Tübingen

Vocabulary (Characteristic)

The characteristic part of the vocabulary for a particular type-theoretic language consists of:

- for every type a, a (possibly empty) set CON_a^L of constants of type a.[3]

Note: It is important to keep constants of type a (c_a) and variables of type a (v_a) appart. A notational difference to predicate logic languages is that the type of a constant or variable can be indicated by an index.

[3] For example, for the type of individuals e we can define an infinite set of constant symbols a, b, c, etc. of this type (just as for predicate logic). Likewise, for the type of a one-place predicate $\langle e, t \rangle$ we can also define an infinite set of one-place first order predicate constants A, B, C, etc. (just as for predicate logic), and so on for every type.

© 2012 Universität Tübingen

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.

© 2012 Universität Tübingen

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.

© 2012 Universität Tübingen

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), (ii), and (iii)
$\forall$xA(x)✓	(i), (ii), and (iv)
Aj x	–
B(A) x	–
$\forall$x$\mathcal{C}$(x) x	–

© 2012 Universität Tübingen

Important: *Expressions* and *Formulas*

Note that in the definitions above, Gamut use the term **expression** instead of *formula* or *sentence* (which were used before in predicate logic). They further specify the difference:

"The inductive definition of the formulas is more complicated than in predicate logic. For what we have to give is a general definition of what is to be an expression of a type $a \in \mathbb{T}$, the **formulas** are then those expressions which are of the particular type *t*."

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

© 2012 Universität Tübingen

Imporant Difference to Predicate Logic

Assume j of type *e* refers to Jumbo, m of type *e* refers to Maya, B is of type $\langle e, \langle e, t \rangle \rangle$, and is reflecting the English two-place predicate "to befriend". The English sentence *Jumbo befriends Maya* would then be translated as:

Predicate Logic: Bjm
Type-Theoretic Logic: (B(m))(j) or alternatively B(m)(j)[5]

Some notable points:

- While the predicate logic formula Bjm combines all three elements B, m, and j together in a single step, in the type-theoretic account, we are bound to strictly **binary functional application**. In the case of two-place predicates, this means that the predicate/function is applied first to one of the arguments, i.e. B(m), and then, in the second step, the outcome is applied to the next argument, i.e. (B(m))(j).
- Note that the latter account reflects the **binary tree structure** posed in (some) syntactic frameworks, whereas the former has a flat structure.
- Also, the **order of the arguments is inverted** here. It is generally assumed that the predicate is first applied to the object of the natural language sentence, and then to the subject.

[5] We leave away the outermost brackets here. Gamut (1991), p. 82 suggest to also drop the brackets around B(m) and hence get the alternative simplified writing B(m)(j).

© 2012 Universität Tübingen

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Faculty of Philosophy
General Linguistics

Section 5: Semantics of Type-Theoretic Languages

Truth Valuation

As seen before for other logical languages, the semantic side of type-theoretic languages consists of the **valuation of truth** given a syntactically valid expression.

Example

Assume *walks* is represented by W of type $\langle e, t \rangle$. Further, *Jumbo* is represented by the constant j of type *e*. We thus have a valid truth-theoretic expression W(j) of type *t* (i.e. a formula) representing *Jumbo walks*.

To evaluate the truth of W(j) we need to define a set of all relevant entities D (the domain) with members d, and a subset $W \subseteq D$ whose members can be said to *walk* (i.e. j $\in$ W). We then define an interpretation function I for which it holds that:

$$I(W)(d) = 1 \text{ iff } d \in W; \text{ and } I(W)(d) = 0 \text{ iff } d \notin W. \quad (1)$$

I(W) is a so-called *characteristic function of W* (over D).

Gamut (1991), Volume 2, p. 83-87.

© 2012 Universität Tübingen

Truth Valuation

However, the complexity of defining interpretation functions for all kinds of different types of expressions (see table below from Gamut) is beyond the scope of this course.

Gamut (1991), Volume 2, p. 86.

Type	**Interpretation**
e	Entity
$\langle e, t\rangle$	Function from entities to truth values, i.e. characteristic function
$\langle e, \langle e, t\rangle\rangle$	Function from entities to sets of entities
$\langle e, \langle e, \langle e, t\rangle\rangle\rangle$	Function from entities to functions from entities to sets of entities
t	Truth value
$\langle t, t\rangle$	Function from truth values to truth values
$\langle e, e\rangle$	Function from entities to entities
$\langle\langle e, t\rangle, \langle e, t\rangle\rangle$	Function from sets of entities to sets of entities
$\langle\langle e, t\rangle, t\rangle$	Characteristic function of a set of sets of entities
$\langle\langle e, t\rangle, \langle\langle e, t\rangle, t\rangle\rangle$	Function from sets of entities to sets of sets of entities
etc.	etc.

© 2012 Universität Tübingen

Summary

Summary

- The theory of types enables us to assign a **type** to any kind of **natural language structure**.
- **Functional applications** of expressions of certain types to one another then enable us to represent the rich **combinatoriality** of natural language structures.
- **Truth valuations** are possible via particular interpretation functions defined for different types of expressions.

© 2012 Universität Tübingen

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.

Kearns, Kate (2011). *Semantics*. New York/London: Palgrave Macmillan.

Kroeger, Paul R. (2019). *Analyzing meaning. An introduction to semantics and pragmatics.* Second corrected and slightly revised version. Berlin: Language Science Press.

Zimmermann, Thomas E. & Sternefeld, Wolfgang (2013). *Introduction to semantics. An essential guide to the composition of meaning.* Mouton de Gruyter.

© 2012 Universität Tübingen

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

© 2012 Universität Tübingen