Formale Semantik
07. Einfach getypte höherstufige L-Sprachen

Roland Schäfer

Institut für Germanistische Sprachwissenschaft
Friedrich-Schiller-Universität Jena

Folien in Überarbeitung. Englische Teile (ab Woche 6) sind noch von 2007!
Stets aktuelle Fassungen: `https://github.com/rsling/VL-Semantik`

Inhalt

Preliminaries

Montague and the generative tradition

- Chierchia & McConnell-Ginet, Heim & Kratzer, etc.: GB-ish semantics

Montague and the generative tradition

- Chierchia & McConnell-Ginet, Heim & Kratzer, etc.: GB-ish semantics
- both syntax and LF in phrase structures

Montague and the generative tradition

- Chierchia & McConnell-Ginet, Heim & Kratzer, etc.: GB-ish semantics
- both syntax and LF in phrase structures
- LF as a proper linguistic level of representation

Montague and the generative tradition

- Chierchia & McConnell-Ginet, Heim & Kratzer, etc.: GB-ish semantics
- both syntax and LF in phrase structures
- LF as a proper linguistic level of representation
- Montague: direct translation of NL into logic

Montague and the generative tradition

- Chierchia & McConnell-Ginet, Heim & Kratzer, etc.: GB-ish semantics
- both syntax and LF in phrase structures
- LF as a proper linguistic level of representation
- Montague: direct translation of NL into logic
- Monatgue's LF is just a notational system for NL semantics

Targets for this week

- Learn to tell the difference between the montagovian and generative approach.

Targets for this week

- Learn to tell the difference between the montagovian and generative approach.
- See the advantage of a general theory of typed languages.

Targets for this week

- Learn to tell the difference between the montagovian and generative approach.
- See the advantage of a general theory of typed languages.
- Understand how λ languages allow dramatically elegant formalizations.

Targets for this week

- Learn to tell the difference between the montagovian and generative approach.
- See the advantage of a general theory of typed languages.
- Understand how λ languages allow dramatically elegant formalizations.
- … while keeping in mind that these devices are extensions to our PC representation for NL semantics.

Denotations

- denotations in set/function-theoretic terms

Denotations

- denotations in set/function-theoretic terms
- a characteristic function (CF) $\mathcal{S}$ of a set S:
  $\mathcal{S}(a) = 1$ *iff* $a \in S$, *else* 0

Denotations

- denotations in set/function-theoretic terms
- a characteristic function (CF) $\mathcal{S}$ of a set S:
  $\mathcal{S}(a) = 1$ *iff* $a \in S$, *else* 0
- a CF 'checks' individuals into a set

Denotations

- denotations in set/function-theoretic terms
- a characteristic function (CF) $\mathcal{S}$ of a set S:
  $\mathcal{S}(a) = 1$ *iff* $a \in S$, *else* 0
- a CF 'checks' individuals into a set
- denotations can be stated as sets or their CF

Generalizing combinatory semantic operations

- interpretation for [$_S$ NP VP]:
  $[\![\, [_S\ NP\ VP]]\!]^{\mathcal{M},g} = 1$ *iff* $[\![NP]\!]^{\mathcal{M},g} \in [\![VP]\!]^{\mathcal{M},g}$

Generalizing combinatory semantic operations

- interpretation for [$_S$ NP VP]:
  $[\![\,[_S\ \mathit{NP}\ \mathit{VP}]\,]\!]^{\mathcal{M},g} = 1$ *iff* $[\![\mathit{NP}]\!]^{\mathcal{M},g} \in [\![\mathit{VP}]\!]^{\mathcal{M},g}$
- Montague generally used CF's in definitions

Generalizing combinatory semantic operations

- interpretation for [$_S$ NP VP]:
  $[\![\, [_S\ NP\ VP]]\!]^{\mathcal{M},g} = 1$ *iff* $[\![NP]\!]^{\mathcal{M},g} \in [\![VP]\!]^{\mathcal{M},g}$
- Montague generally used CF's in definitions
- evaluating [$_S$ [$_{NP}$ *Mary*] [$_{VP}$ *sleeps*]] as a matter of functional application (FA):

Generalizing combinatory semantic operations

- interpretation for [$_S$ NP VP]:
  $[\![[_S \textit{NP VP}]]\!]^{\mathcal{M},g} = 1$ *iff* $[\![\textit{NP}]\!]^{\mathcal{M},g} \in [\![\textit{VP}]\!]^{\mathcal{M},g}$
- Montague generally used CF's in definitions
- evaluating [$_S$ [$_{NP}$ *Mary*] [$_{VP}$ *sleeps*]] as a matter of functional application (FA):
  - $[\![\textit{Mary}]\!]^{\mathcal{M},g} = \textit{Mary in } \mathcal{M}$

- interpretation for [$_S$ NP VP]:
  $[\![[_S \mathit{NP}\ \mathit{VP}]]\!]^{\mathcal{M},g} = 1$ *iff* $[\![\mathit{NP}]\!]^{\mathcal{M},g} \in [\![\mathit{VP}]\!]^{\mathcal{M},g}$
- Montague generally used CF's in definitions
- evaluating [$_S$ [$_{NP}$ *Mary*] [$_{VP}$ *sleeps*]] as a matter of functional application (FA):
  - $[\![\mathit{Mary}]\!]^{\mathcal{M},g} = $ *Mary in* $\mathcal{M}$
  - $[\![\mathit{sleeps}]\!]^{\mathcal{M},g}$ *be the CF of the set of sleepers in* $\mathcal{M}$

Generalizing combinatory semantic operations

- interpretation for [$_S$ NP VP]:
  $[\![[_S NP\ VP]]\!]^{\mathcal{M},g} = 1$ *iff* $[\![NP]\!]^{\mathcal{M},g} \in [\![VP]\!]^{\mathcal{M},g}$
- Montague generally used CF's in definitions
- evaluating [$_S$ [$_{NP}$ *Mary*] [$_{VP}$ *sleeps*]] as a matter of functional application (FA):
  - $[\![Mary]\!]^{\mathcal{M},g} =$ *Mary in* $\mathcal{M}$
  - $[\![sleeps]\!]^{\mathcal{M},g}$ *be the CF of the set of sleepers in* $\mathcal{M}$
  - $[\![S]\!]^{\mathcal{M},g} = [\![sleeps]\!]^{\mathcal{M},g}([\![Mary]\!]^{\mathcal{M},g})$

Generalizing combinatory semantic operations

- interpretation for [$_{S}$ NP VP]:
  $[\![\,[_S\ \mathit{NP}\ \mathit{VP}]\,]\!]^{\mathcal{M},g} = 1$ *iff* $[\![\mathit{NP}]\!]^{\mathcal{M},g} \in [\![\mathit{VP}]\!]^{\mathcal{M},g}$
- Montague generally used CF's in definitions
- evaluating [$_{S}$ [$_{NP}$ *Mary*] [$_{VP}$ *sleeps*]] as a matter of functional application (FA):
  - $[\![\mathit{Mary}]\!]^{\mathcal{M},g} =$ *Mary in* $\mathcal{M}$
  - $[\![\mathit{sleeps}]\!]^{\mathcal{M},g}$ *be the CF of the set of sleepers in* $\mathcal{M}$
  - $[\![S]\!]^{\mathcal{M},g} = [\![\mathit{sleeps}]\!]^{\mathcal{M},g}([\![\mathit{Mary}]\!]^{\mathcal{M},g})$
  - ideally: generalize to all nodes

The superscript notation

- all functions from S_1 to S_2

The superscript notation

- all functions from S_1 to S_2
- $S_2^{S_1}$

The superscript notation

- all functions from S_1 to S_2
- $S_2^{S_1}$
- for $T = \{0, 1\}$

The superscript notation

- all functions from S_1 to S_2
- $S_2^{S_1}$
- for $T = \{0, 1\}$
  - T^D: all pred_1
-

- all functions from S_1 to S_2
- $S_2^{S_1}$
- for $T = \{0, 1\}$
  - T^D: all pred_1
  - $T^{D \times D}$: all pred_2
-

Simply typed languages

Some new names

- base for Dowty et al.: L_1, a first-order predicate language as we know it

Some new names

- base for Dowty et al.: L_1, a first-order predicate language as we know it
- semantic renaming of types:

Some new names

- base for Dowty et al.: L_1, a first-order predicate language as we know it
- semantic renaming of types:
  - terms: $\langle e \rangle$ (entity-denoting)

Some new names

- base for Dowty et al.: L_1, a first-order predicate language as we know it
- semantic renaming of types:
  - terms: $\langle e \rangle$ (entity-denoting)
  - formulas: $\langle t \rangle$ (truth-valued)

Some new names

- base for Dowty et al.: L_1, a first-order predicate language as we know it
- semantic renaming of types:
  - terms: $\langle e \rangle$ (entity-denoting)
  - formulas: $\langle t \rangle$ (truth-valued)
  - pred_1: $\langle e, t \rangle$

Some new names

- base for Dowty et al.: L_1, a first-order predicate language as we know it
- semantic renaming of types:
  - terms: $\langle e \rangle$ (entity-denoting)
  - formulas: $\langle t \rangle$ (truth-valued)
  - pred_1: $\langle e, t \rangle$
  - pred_2: $\langle e, \langle e, t \rangle \rangle$

Possible denotations of types

- D_α possible denotation (a set) of expressions of type α

Possible denotations of types

- D_α possible denotation (a set) of expressions of type α
- $D_{\langle e \rangle} = U$ (Dowty et al.'s A)

Possible denotations of types

- D_α possible denotation (a set) of expressions of type α
- $D_{\langle e \rangle} = U$ (Dowty et al.'s A)
- $D_{\langle t \rangle} = \{0, 1\}$

Possible denotations of types

- D_α possible denotation (a set) of expressions of type α
- $D_{\langle e \rangle} = U$ (Dowty et al.'s A)
- $D_{\langle t \rangle} = \{0, 1\}$
- recursively: $D_{\langle \alpha, \beta \rangle} = D_{\langle \beta \rangle}^{D_{\langle \alpha \rangle}}$

Possible denotations of types

- D_α possible denotation (a set) of expressions of type α
- $D_{\langle e \rangle} = U$ (Dowty et al.'s A)
- $D_{\langle t \rangle} = \{0, 1\}$
- recursively: $D_{\langle \alpha, \beta \rangle} = D_{\langle \beta \rangle}^{D_{\langle \alpha \rangle}}$
- e.g., $D_{\langle e, t \rangle} = D_{\langle t \rangle}^{D_{\langle e \rangle}}$

Possible denotations of types

- D_α possible denotation (a set) of expressions of type α
- $D_{\langle e \rangle} = U$ (Dowty et al.'s A)
- $D_{\langle t \rangle} = \{0, 1\}$
- recursively: $D_{\langle \alpha, \beta \rangle} = D_{\langle \beta \rangle}^{D_{\langle \alpha \rangle}}$
- e.g., $D_{\langle e,t \rangle} = D_{\langle t \rangle}^{D_{\langle e \rangle}}$
- $D_{\langle e, \langle e,t \rangle \rangle} = (D_{\langle t \rangle}^{D_{\langle e \rangle}})^{D_{\langle e \rangle}}$

Possible denotations of types

- D_α possible denotation (a set) of expressions of type α
- $D_{\langle e \rangle} = U$ (Dowty et al.'s A)
- $D_{\langle t \rangle} = \{0, 1\}$
- recursively: $D_{\langle \alpha, \beta \rangle} = D_{\langle \beta \rangle}^{D_{\langle \alpha \rangle}}$
- e.g., $D_{\langle e,t \rangle} = D_{\langle t \rangle}^{D_{\langle e \rangle}}$
- $D_{\langle e, \langle e,t \rangle \rangle} = (D_{\langle t \rangle}^{D_{\langle e \rangle}})^{D_{\langle e \rangle}}$
- just a systematic way of naming types, model-theoretic interpretations still by V, g

Defining types

- in our PS syntax: S as start symbol

Defining types

- in our PS syntax: S as start symbol
- in the typed system: sentences should be of type $\langle t \rangle$

Defining types

- in our PS syntax: S as start symbol
- in the typed system: sentences should be of type $\langle t \rangle$
- complex types: functions from $\langle e \rangle$ to $\langle t \rangle$
  or generally from any (complex) type to any (comlex) type

Complex types as functions

- saturation of complex types by FA:

Complex types as functions

- saturation of complex types by FA:
  - γ *is of type* $\langle e, \langle e, t \rangle \rangle$, δ *of* $\langle e, t \rangle$, α *and* β *of* $\langle e \rangle$

Complex types as functions

- saturation of complex types by FA:
  - γ *is of type* $\langle e, \langle e, t\rangle\rangle$, δ *of* $\langle e, t\rangle$, α *and* β *of* $\langle e\rangle$
  - then $\gamma(\alpha)$ *is of type* $\langle e, t\rangle$

Complex types as functions

- saturation of complex types by FA:
  - γ *is of type* $\langle e, \langle e, t \rangle \rangle$, δ *of* $\langle e, t \rangle$, α *and* β *of* $\langle e \rangle$
  - then $\gamma(\alpha)$ *is of type* $\langle e, t \rangle$
  - and $\delta(\beta)$ *is of type* $\langle t \rangle$

Complex types as functions

- saturation of complex types by FA:
  - γ *is of type* $\langle e, \langle e, t\rangle\rangle$, δ *of* $\langle e, t\rangle$, α *and* β *of* $\langle e\rangle$
  - then $\gamma(\alpha)$ *is of type* $\langle e, t\rangle$
  - and $\delta(\beta)$ *is of type* $\langle t\rangle$
- for any pred$_2$ P and its arguments a_1, a_2, $P(a_2)(a_1)$ is a wff

Complex types as functions

- saturation of complex types by FA:
  - γ *is of type* $\langle e, \langle e, t\rangle\rangle$, δ *of* $\langle e, t\rangle$, α *and* β *of* $\langle e\rangle$
  - then $\gamma(\alpha)$ *is of type* $\langle e, t\rangle$
  - and $\delta(\beta)$ *is of type* $\langle t\rangle$
- for any pred₂ P and its arguments a_1, a_2, $P(a_2)(a_1)$ is a wff
- connectives are of types $\langle t, t\rangle$ ($\neg$), $\langle t, \langle t, t\rangle\rangle$ ($\wedge$, etc.)

General semantics of typed languages

- generalized CF/FA approach

General semantics of typed languages

- generalized CF/FA approach
- $\langle e \rangle$-types (terms):
  $[\![a_n]\!]^{\mathcal{M},g} = V(a_n)$
  $[\![x_n]\!]^{\mathcal{M},g} = g(x_n)$

General semantics of typed languages

- generalized CF/FA approach
- $\langle e \rangle$-types (terms):
  $[\![a_n]\!]^{\mathcal{M},g} = V(a_n)$
  $[\![x_n]\!]^{\mathcal{M},g} = g(x_n)$
- the rest: functional application
  $[\![\delta(\alpha)]\!]^{\mathcal{M},g} = [\![\delta]\!]^{\mathcal{M},g}([\![\alpha]\!]^{\mathcal{M},g})$

Refinement

- *Type* is the set of types

Refinement

- *Type* is the set of types
- recursively defined complex types $\langle a, b\rangle$: infinite

Refinement

- *Type* is the set of types
- recursively defined complex types $\langle a, b \rangle$: infinite
- type label $\langle \alpha \rangle$

Refinement

- *Type* is the set of types
- recursively defined complex types $\langle a, b \rangle$: infinite
- type label $\langle \alpha \rangle$
- vs. set of meaningful expressions of that type: $ME_{\langle \alpha \rangle}$

Higher order

- first order languages: variables over individuals ($\langle e \rangle$-types)

Higher order

- first order languages: variables over individuals ($\langle e \rangle$-types)
- n-order: variables over higher types ($\langle e, t \rangle$-types etc.)

Higher order

- first order languages: variables over individuals ($\langle e \rangle$-types)
- n-order: variables over higher types ($\langle e, t \rangle$-types etc.)
- $P_{\langle e,t \rangle}$ or $Q_{\langle e,\langle e,t \rangle\rangle}$: constants of higher types

Higher order

- first order languages: variables over individuals ($\langle e \rangle$-types)
- n-order: variables over higher types ($\langle e, t \rangle$-types etc.)
- $P_{\langle e,t \rangle}$ or $Q_{\langle e,\langle e,t \rangle \rangle}$: constants of higher types
- so: $v_{1_{\langle e,t \rangle}} [v_1(m)]$

Higher order

- first order languages: variables over individuals ($\langle e \rangle$-types)
- n-order: variables over higher types ($\langle e, t \rangle$-types etc.)
- $P_{\langle e,t \rangle}$ or $Q_{\langle e,\langle e,t \rangle \rangle}$: constants of higher types
- so: $v_{1_{\langle e,t \rangle}} [v_1(m)]$
- if $V(m) = Mary$, v_1 is the set of all of Mary's properties

Typing variables

- we write:

Typing variables

- we write:
  - $v_{n_{\langle\alpha\rangle}}$ for the n-th variable of type $\langle\alpha\rangle$

Typing variables

- we write:
  - $v_{n_{\langle\alpha\rangle}}$ for the n-th variable of type $\langle\alpha\rangle$
  - Dowty et al.: $v_{n,\langle\alpha\rangle}$

Typing variables

- we write:
  - $v_{n_{\langle\alpha\rangle}}$ for the n-th variable of type $\langle\alpha\rangle$
  - Dowty et al.: $v_{n,\langle\alpha\rangle}$
- alternatively abbreviated by old symbols x_1, a, P, etc.

Constants, variables, functions

- non-logical constant α: $[\![\alpha]\!]^{\mathcal{M},g} = V(\alpha)$

Constants, variables, functions

- non-logical constant α: $[\![\alpha]\!]^{\mathcal{M},g} = V(\alpha)$
- variable α: $[\![\alpha]\!]^{\mathcal{M},g} = V(\alpha)$

Constants, variables, functions

- non-logical constant α: $[\![\alpha]\!]^{\mathcal{M},g} = V(\alpha)$
- variable α: $[\![\alpha]\!]^{\mathcal{M},g} = V(\alpha)$
- $\alpha \in \langle a, b \rangle,\ \beta \in a$, then $[\![\alpha(\beta)]\!]^{\mathcal{M},g} = [\![\alpha]\!]^{\mathcal{M},g}([\![\beta]\!]^{\mathcal{M},g})$

- logical constants interpreted as functions in $\{0,1\}$ as usual

Logical constants and quantifiers

- logical constants interpreted as functions in {0,1} as usual
- if $v_{1_{\langle\alpha\rangle}}$ is a variable and $\phi \in ME_t$
  then $[\![(\forall v_1)\phi]\!]^{\mathcal{M},g} = 1$ *iff*
  for all $a \in D_\alpha$ $[\![\phi]\!]^{\mathcal{M},g[a/v_1]} = 1$

An example

- quantified variable of type $\langle e, t\rangle$: $v_{0_{\langle e,t\rangle}}$

An example

- quantified variable of type $\langle e,t\rangle$: $v_{0_{\langle e,t\rangle}}$
- $\forall v_{0_{\langle e,t\rangle}}\left[v_{0_{\langle e,t\rangle}}(j) \rightarrow v_{0_{\langle e,t\rangle}}(d)\right]$

An example

- quantified variable of type $\langle e,t\rangle$: $v_{0_{\langle e,t\rangle}}$
- $\forall v_{0_{\langle e,t\rangle}}\left[v_{0_{\langle e,t\rangle}}(j) \rightarrow v_{0_{\langle e,t\rangle}}(d)\right]$
- for $j, d \in ME_{\langle e\rangle}$

An example

- quantified variable of type $\langle e,t\rangle$: $v_{0_{\langle e,t\rangle}}$
- $\forall v_{0_{\langle e,t\rangle}} \left[v_{0_{\langle e,t\rangle}}(j) \rightarrow v_{0_{\langle e,t\rangle}}(d)\right]$
- for $j, d \in ME_{\langle e\rangle}$
- one property of every individual: being alone in its union set

An example

- quantified variable of type $\langle e,t\rangle$: $v_{0_{\langle e,t\rangle}}$
- $\forall v_{0_{\langle e,t\rangle}}\left[v_{0_{\langle e,t\rangle}}(j) \rightarrow v_{0_{\langle e,t\rangle}}(d)\right]$
- for $j, d \in ME_{\langle e\rangle}$
- one property of every individual: being alone in its union set
- hence, $j = d$

An example

- quantified variable of type $\langle e, t\rangle$: $v_{0_{\langle e,t\rangle}}$
- $\forall v_{0_{\langle e,t\rangle}} \left[v_{0_{\langle e,t\rangle}}(j) \rightarrow v_{0_{\langle e,t\rangle}}(d)\right]$
- for $j, d \in ME_{\langle e\rangle}$
- one property of every individual: being alone in its union set
- hence, $j = d$
- else in $\forall v_{0_{\langle e,t\rangle}}$, $\forall$ wouldn't hold

Defining *non*

- productive adjectival prefix: *non-adjacent*, *non-local*, etc.

Defining *non*

- productive adjectival prefix: *non-adjacent*, *non-local*, etc.
- inverting the characteristic function of the adjective

Defining *non*

- productive adjectival prefix: *non-adjacent*, *non-local*, etc.
- inverting the characteristic function of the adjective
- result denotes complement of the original adjective in $D_{\langle e \rangle}$

Defining *non*

- productive adjectival prefix: *non-adjacent*, *non-local*, etc.
- inverting the characteristic function of the adjective
- result denotes complement of the original adjective in $D_{\langle e \rangle}$
- *adjective*: $\langle e, t \rangle$, *non*: $\langle \langle e, t \rangle, \langle e, t \rangle \rangle$

Defining *non*

- productive adjectival prefix: *non-adjacent*, *non-local*, etc.
- inverting the characteristic function of the adjective
- result denotes complement of the original adjective in $D_{\langle e \rangle}$
- *adjective*: $\langle e, t\rangle$, *non*: $\langle\langle e, t\rangle, \langle e, t\rangle\rangle$
- a function h s.t. for every $k \in D_{\langle e,t \rangle}$ and every $d \in D_{\langle e \rangle}$
  $(h(k))(d) = 1$ iff $k(d) = 0$ and
  $(h(k))(d) = 0$ iff $k(d) = 1$

Argument deletion

- understood objects in: *I eat.* - *Vanity kills.* - etc.

Argument deletion

- understood objects in: *I eat.* - *Vanity kills.* - etc.
- *eat* is in $ME_{\langle e,\langle e,t\rangle\rangle}$

Argument deletion

- understood objects in: *I eat.* - *Vanity kills.* - etc.
- *eat* is in $ME_{\langle e,\langle e,t\rangle\rangle}$
- assume a silent logical constant: R_O in $ME_{\langle\langle e,\langle e,t\rangle\rangle,\langle e,t\rangle\rangle}$

Argument deletion

- understood objects in: *I eat.* - *Vanity kills.* - etc.
- *eat* is in $ME_{\langle e,\langle e,t\rangle\rangle}$
- assume a silent logical constant: R_O in $ME_{\langle\langle e,\langle e,t\rangle\rangle,\langle e,t\rangle\rangle}$
- a function h s.t. for all $k \in D_{\langle e,\langle e,t\rangle\rangle}$ and all $d \in D_{\langle e\rangle}$ $h(k)(d) = 1$ iff there is some $d' \in D_{\langle e\rangle}$ s.t. k(d′)(d)=1

Argument deletion

- understood objects in: *I eat.* - *Vanity kills.* - etc.
- *eat* is in $ME_{\langle e,\langle e,t\rangle\rangle}$
- assume a silent logical constant: R_O in $ME_{\langle\langle e,\langle e,t\rangle\rangle,\langle e,t\rangle\rangle}$
- a function h s.t. for all $k \in D_{\langle e,\langle e,t\rangle\rangle}$ and all $d \in D_{\langle e\rangle}$
  $h(k)(d) = 1$ iff there is some $d' \in D_{\langle e\rangle}$ s.t. k(d′)(d)=1
- passives as similar subject deletion

Lambda languages

All there is to λ

- a new variable binder

All there is to λ

- a new variable binder
- allows abstraction over wff's of arbitrary complexity

All there is to λ

- a new variable binder
- allows abstraction over wff's of arbitrary complexity
- similar to $\{x \| \phi\}$ (read as 'the set of all x s.t. ϕ')

All there is to λ

- a new variable binder
- allows abstraction over wff's of arbitrary complexity
- similar to $\{x \| \phi\}$ (read as 'the set of all x s.t. ϕ')
- we get $\lambda x\,[\phi]$

All there is to λ

- a new variable binder
- allows abstraction over wff's of arbitrary complexity
- similar to $\{x \| \phi\}$ (read as 'the set of all x s.t. ϕ')
- we get $\lambda x\,[\phi]$
- on Montague's typewriter: $\hat{x}\,[\phi]$

All there is to λ

- a new variable binder
- allows abstraction over wff's of arbitrary complexity
- similar to $\{x \| \phi\}$ (read as 'the set of all x s.t. ϕ')
- we get $\lambda x\,[\phi]$
- on Montague's typewriter: $\hat{x}\,[\phi]$
- does not create a set but a function which can be taken as the CF of a set

λ abstraction

- for every wff ϕ, any $x \in Var$, and any $a \in Con$

λ abstraction

- for every wff ϕ, any $x \in Var$, and any $a \in Con$
- λ abstraction: $\phi \rightarrow \lambda x \left[\phi^{[a/x]}\right](a)$

λ abstraction

- for every wff ϕ, any $x \in Var$, and any $a \in Con$
- λ abstraction: $\phi \to \lambda x \left[\phi^{[a/x]}\right](a)$
- read $\phi^{(a/x)}$ as '*phi* with every a replaced by x'

λ abstraction

- for every wff ϕ, any $x \in Var$, and any $a \in Con$
- λ abstraction: $\phi \rightarrow \lambda x \left[\phi^{[a/x]}\right](a)$
- read $\phi^{(a/x)}$ as '*phi* with every *a* replaced by *x*'
- *x* can be of any type

Two informal examples

- $\lambda x_{\langle e \rangle}\ [L(x)]$ is the characteristic function of the set of those individuals $d \in D_{\langle e \rangle}$ which have property L

Two informal examples

- $\lambda x_{\langle e \rangle}\ [L(x)]$ is the characteristic function of the set of those individuals $d \in D_{\langle e \rangle}$ which have property L
- $\lambda x_{\langle e,t \rangle}\ [x(l)]$ is the characteristic function of the set of those properties $k \in D_{\langle e,t \rangle}$ that the individual l has

λ conversion

- $\lambda x\,[L(x)]$ is the abstract of $L(a)$ (with some individual a)

λ conversion

- $\lambda x\,[L(x)]$ is the abstract of $L(a)$ (with some individual a)
- hence, it holds: $\lambda x\,[L(x)]\,(a) \Leftrightarrow L(a)$

λ conversion

- $\lambda x\,[L(x)]$ is the abstract of $L(a)$ (with some individual a)
- hence, it holds: $\lambda x\,[L(x)]\,(a) \Leftrightarrow L(a)$
- for every wff ϕ, any $x \in \mathit{Var}$, and any $a \in \mathit{Con}$

λ conversion

- $\lambda x\,[L(x)]$ is the abstract of $L(a)$ (with some individual a)
- hence, it holds: $\lambda x\,[L(x)]\,(a) \Leftrightarrow L(a)$
- for every wff ϕ, any $x \in Var$, and any $a \in Con$
- λ conversion: $\lambda x\,[\phi]\,(a) \rightarrow \phi^{[x/a]}$

λ in and out

- $\lambda x\,[\phi]\,(a) \leftrightarrow \phi^{[x/a]}$

λ in and out

- $\lambda x\,[\phi]\,(a) \leftrightarrow \phi^{[x/a]}$
- not just syntactically, since truth conditions are equivalent

λ in and out

- $\lambda x \left[\phi\right] (a) \leftrightarrow \phi^{[x/a]}$
- not just syntactically, since truth conditions are equivalent
- $\lambda x \left[\phi\right] (a) \Leftrightarrow \phi^{[x/a]}$

λ in and out

- $\lambda x\,[\phi]\,(a) \leftrightarrow \phi^{[x/a]}$
- not just syntactically, since truth conditions are equivalent
- $\lambda x\,[\phi]\,(a) \Leftrightarrow \phi^{[x/a]}$
- notice: $\lambda x_{\langle\alpha\rangle}\,[\phi]$ is in $ME_{\langle\alpha,t\rangle}$

λ in and out

- $\lambda x\,[\phi]\,(a) \leftrightarrow \phi^{[x/a]}$
- not just syntactically, since truth conditions are equivalent
- $\lambda x\,[\phi]\,(a) \Leftrightarrow \phi^{[x/a]}$
- notice: $\lambda x_{\langle\alpha\rangle}\,[\phi]$ is in $ME_{\langle\alpha,t\rangle}$
- while ϕ (as a wff) is in $ME_{\langle t\rangle}$

The full rules

- Dowty et al., 102f. (*Syn C.10* and *Sem 10*)

The full rules

- Dowty et al., 102f. (*Syn C.10* and *Sem 10*)
- If $\alpha \in ME_a$ and $u \in Var_b$, then $\lambda u\,[\alpha] \in ME_{\langle b,a \rangle}$.

The full rules

- Dowty et al., 102f. (*Syn C.10* and *Sem 10*)
- If $\alpha \in ME_\alpha$ and $u \in Var_b$, then $\lambda u\,[\alpha] \in ME_{\langle b,a\rangle}$.
- If $\alpha \in ME_a$ and $u \in Var_b$ then $[\![\lambda u\,[\alpha]]\!]^{\mathcal{M},g}$ is that function h from D_b into D_a s.t. for all objects k in D_b, $h(k)$ is equal to $[\![\alpha]\!]^{\mathcal{M},g[k/u]}$.

The *non* example revised (Dowty et al., 104)

- $\forall x \forall v_{0\langle e,t\rangle} \left[(\mathbf{non}(v_{0_{\langle e,t\rangle}}))(x) \leftrightarrow \neg(v_{0_{\langle e,t\rangle}}(x)) \right]$

The *non* example revised (Dowty et al., 104)

- $\forall x \forall v_{0\langle e,t\rangle} \left[(\mathbf{non}(v_{0_{\langle e,t\rangle}}))(x) \leftrightarrow \neg(v_{0_{\langle e,t\rangle}}(x)) \right]$
- $\forall v_{0_{\langle e,t\rangle}} \left[\lambda x \left[(\mathbf{non}(v_{0_{\langle e,t\rangle}}))(x) \right] = \lambda x \left[\neg(v_{0_{\langle e,t\rangle}}(x)) \right] \right]$

The *non* example revised (Dowty et al., 104)

- $\forall x \forall v_{0\langle e,t\rangle} \left[(\mathbf{non}(v_{0_{\langle e,t\rangle}}))(x) \leftrightarrow \neg(v_{0_{\langle e,t\rangle}}(x)) \right]$
- $\forall v_{0_{\langle e,t\rangle}} \left[\lambda x \left[(\mathbf{non}(v_{0_{\langle e,t\rangle}}))(x) \right] = \lambda x \left[\neg(v_{0_{\langle e,t\rangle}}(x)) \right] \right]$
- $\forall v_{0_{\langle e,t\rangle}} \left[\mathbf{non}(v_{0_{\langle e,t\rangle}}) = \lambda x \left[\neg(v_{0_{\langle e,t\rangle}}(x)) \right] \right]$
  (since $\lambda x\, [\mathbf{non}(v)(x)]$ is unnecessarily abstract/η reduction)

The *non* example revised (Dowty et al., 104)

- $\forall x \forall v_{0_{\langle e,t \rangle}} \left[(\mathbf{non}(v_{0_{\langle e,t \rangle}}))(x) \leftrightarrow \neg(v_{0_{\langle e,t \rangle}}(x)) \right]$
- $\forall v_{0_{\langle e,t \rangle}} \left[\lambda x \left[(\mathbf{non}(v_{0_{\langle e,t \rangle}}))(x) \right] = \lambda x \left[\neg(v_{0_{\langle e,t \rangle}}(x)) \right] \right]$
- $\forall v_{0_{\langle e,t \rangle}} \left[\mathbf{non}(v_{0_{\langle e,t \rangle}}) = \lambda x \left[\neg(v_{0_{\langle e,t \rangle}}(x)) \right] \right]$
  (since $\lambda x\, [\mathbf{non}(v)(x)]$ is unnecessarily abstract/η reduction)
- $\lambda v_{0_{\langle e,t \rangle}} \left[\mathbf{non}(v_{0_{\langle e,t \rangle}}) = \lambda v_{0_{\langle e,t \rangle}} \left[\lambda x \left[\neg(v_{0_{\langle e,t \rangle}}(x)) \right] \right] \right]$

The *non* example revised (Dowty et al., 104)

- $\forall x \forall v_{0\langle e,t \rangle} \left[(\mathbf{non}(v_{0_{\langle e,t \rangle}}))(x) \leftrightarrow \neg(v_{0_{\langle e,t \rangle}}(x)) \right]$
- $\forall v_{0_{\langle e,t \rangle}} \left[\lambda x \left[(\mathbf{non}(v_{0_{\langle e,t \rangle}}))(x) \right] = \lambda x \left[\neg(v_{0_{\langle e,t \rangle}}(x)) \right] \right]$
- $\forall v_{0_{\langle e,t \rangle}} \left[\mathbf{non}(v_{0_{\langle e,t \rangle}}) = \lambda x \left[\neg(v_{0_{\langle e,t \rangle}}(x)) \right] \right]$

  (since $\lambda x [\mathbf{non}(v)(x)]$ is unnecessarily abstract/η reduction)
- $\lambda v_{0_{\langle e,t \rangle}} \left[\mathbf{non}(v_{0_{\langle e,t \rangle}}) = \lambda v_{0_{\langle e,t \rangle}} \left[\lambda x \left[\neg(v_{0_{\langle e,t \rangle}}(x)) \right] \right] \right]$
- and since that is about all assignments for $\lambda v_{0_{\langle e,t \rangle}}$:

  $\mathbf{non} = \lambda v_{0_{\langle e,t \rangle}} \left[\lambda x \left[\neg v_{0_{\langle e,t \rangle}}(x) \right] \right]$

Mary is non-adjacent.
(translate ‘adjacent’ as $c_{0_{\langle e,t\rangle}}$, ‘Mary’ as $c_{0_{\langle e\rangle}}$, ignore the copula)

The behavior of quantified NPs

- syntactically like referential NPs

The behavior of quantified NPs

- syntactically like referential NPs
- semantically like PC quantifiers

The behavior of quantified NPs

- syntactically like referential NPs
- semantically like PC quantifiers
- *Every student walks.*: $\forall v_{0_{\langle e \rangle}} \left[c_{0_{\langle e,t \rangle}} (v_{0_{\langle e \rangle}}) \rightarrow c_{1_{\langle e,t \rangle}} (v_{0_{\langle e \rangle}}) \right]$

The behavior of quantified NPs

- syntactically like referential NPs
- semantically like PC quantifiers
- *Every student walks.*: $\forall v_{0_{\langle e \rangle}} \left[c_{0_{\langle e,t \rangle}}(v_{0_{\langle e \rangle}}) \rightarrow c_{1_{\langle e,t \rangle}}(v_{0_{\langle e \rangle}}) \right]$
- *Some student walks.*: $\forall v_{0_{\langle e \rangle}} \left[c_{0_{\langle e,t \rangle}}(v_{0_{\langle e \rangle}}) \wedge c_{1_{\langle e,t \rangle}}(v_{0_{\langle e \rangle}}) \right]$

The behavior of quantified NPs

- syntactically like referential NPs
- semantically like PC quantifiers
- *Every student walks.*: $\forall v_{0_{\langle e \rangle}} \left[c_{0_{\langle e,t \rangle}}(v_{0_{\langle e \rangle}}) \rightarrow c_{1_{\langle e,t \rangle}}(v_{0_{\langle e \rangle}}) \right]$
- *Some student walks.*: $\forall v_{0_{\langle e \rangle}} \left[c_{0_{\langle e,t \rangle}}(v_{0_{\langle e \rangle}}) \wedge c_{1_{\langle e,t \rangle}}(v_{0_{\langle e \rangle}}) \right]$
- making referential NPs and QNPs the same type?

A higher type

- $\lambda v_{0_{\langle e,t\rangle}} \forall v_{0_{\langle e\rangle}} \left[c_{0_{\langle e,t\rangle}}\left(v_{0_{\langle e\rangle}}\right) \rightarrow v_{0_{\langle e,t\rangle}}\left(v_{0_{\langle e\rangle}}\right) \right]$

A higher type

- $\lambda v_{0_{\langle e,t \rangle}} \forall v_{0_{\langle e \rangle}} \left[c_{0_{\langle e,t \rangle}} \left(v_{0_{\langle e \rangle}} \right) \rightarrow v_{0_{\langle e,t \rangle}} \left(v_{0_{\langle e \rangle}} \right) \right]$
- a second order function

A higher type

- $\lambda v_{0_{\langle e,t \rangle}} \forall v_{0_{\langle e \rangle}} \left[c_{0_{\langle e,t \rangle}}\left(v_{0_{\langle e \rangle}}\right) \rightarrow v_{0_{\langle e,t \rangle}}\left(v_{0_{\langle e \rangle}}\right) \right]$
- a second order function
- characterizes the set of all predicates true of every student

A higher type

- $\lambda v_{0_{\langle e,t\rangle}} \forall v_{0_{\langle e\rangle}} \left[c_{0_{\langle e,t\rangle}} \left(v_{0_{\langle e\rangle}} \right) \rightarrow v_{0_{\langle e,t\rangle}} \left(v_{0_{\langle e\rangle}} \right) \right]$
- a second order function
- characterizes the set of all predicates true of every student
- equally: $\lambda v_{0_{\langle e,t\rangle}} \exists v_{0_{\langle e\rangle}} \left[c_{0_{\langle e,t\rangle}} \left(v_{0_{\langle e\rangle}} \right) \wedge v_{0_{\langle e,t\rangle}} \left(v_{0_{\langle e\rangle}} \right) \right]$

Combining with some predicate

Literatur I

Autor

Kontakt

Prof. Dr. Roland Schäfer
Institut für Germanistische Sprachwissenschaft
Friedrich-Schiller-Universität Jena
Fürstengraben 30
07743 Jena

https://rolandschaefer.net
roland.schaefer@uni-jena.de

Lizenz