Handout 2: Two Type Logics (Muskens 1995, Ch. 2)

Semantics C (Spring 2010)

Montague Grammar:

- a very elegant and a very simple theory of natural language semantics
- its elegance and simplicity are obscured by a needlessly baroque formalization
- as Barwise and Cooper (1981) put it:

  Montague had a certain job that he wanted to do and used whatever tools he had at hand to do it. If the product looks a bit like a Rube Goldberg machine, well, at least it works pretty well.

- among linguists, the Rube Goldberg character of Montague's theory has created a general impression that the subject is 'difficult' – it would be worthwhile to streamline the theory a bit even if this would only serve the purpose of taking away this false impression
- streamlining the theory serves another goal as well: if we cut down the theory to its bare essentials, it will be easier to generalize it afterwards

Montague based his semantic theory on a higher-order logic which he called IL—Intensional Logic.

- he defined this system as an extension of Russell's Simple Theory of Types (Russell (1908)), a logic which had found its classical formulation in Church (1940)
- the great complexity of the logic IL is to be held responsible, at least in part, for Barwise & Cooper's remark, but in fact Montague's semantic theory can be based on much simpler logics
- Gallin (1975) observes that from a formal point of view it is natural to interpret IL in a two-sorted variant of Russell's original type theory, a logic which he called TY_2
- replacing IL by this version of classical higher-order logic already leads to a much smoother set-up of Montague's original system
- we will discuss TY_2 presently

The partialization of Montague's semantic theory in later chapters will not be based on the logic TY_2 – there is a further simplification that we want to make.

- the models of TY_2 are hierarchies built up from certain ground domains by the single rule that the set of all unary functions from one domain to another is itself a domain
- why only *unary* functions? we need functions and relations in more than one argument, but these can be *coded* by unary functions
- two steps are needed to code a multi-argument relation:
  - identify it with its characteristic function, a multi-argument function; this identification is very simple and unobjectionable
  - the second step is highly tricked is based on Schönfinkel's and Curry's observation that there is a one-to-one correspondence between multi-argument functions and certain unary functions of higher type

- e.g., an ordinary three-place relation on individuals like the relation *give* is equated with its characteristic function, which, in its turn, is identified with a function from individuals to functions from individuals to functions from individuals to truth-values
- the virtue of this approach is a certain parsimony: some objects (relations, multi-argument functions) are replaced by others (unary functions) that are needed anyway
- from a purely technical point of view there is nothing that can be said against this move
- the price: an unintuitive complication of our models and an unintuitive recursive encoding of relatively simple objects by relatively complex ones

We will therefore define a second type theory TT_2 in which these unnecessary complications are done away with.

- TT_2 can informally be described as consisting of the syntax of Church's type logic interpreted on the relational models of Orey (1959)
- a comparison between TY_2 and TT_2 will reveal that, although the two systems have different models, the entailment relation on TT_2 is the same as that on the functional logic TY_2; as the latter is a classical logic, this means that TT_2 is classical as well

1 The functional type logic $\mathbf{TY}_2$

An m-sorted functional type logic TY_m:

- has $m+1$ basic or ground types
- one of these types must be the type t, which stands for *truth values*

The two other ground types in Gallin's TY_2:

- s, which stands for *world-time pairs*
- e, which is the type of *(possible) individuals* or *entities* (individuals which exist in some possible world at some point in time)

From these basic types, complex types are built up as follows:

(1) (TY_2 types) The set of TY_2 *types* is the smallest set of strings such that:

  i. e, s and t are TY_2 types;
  ii. if α and β are TY_2 types, then $(\alpha\beta)$ is a TY_2 type.

Types $\alpha\beta$ (we usually omit outer brackets) are associated with functions from objects of type α to objects of type β:

- e.g., $e(e(et))$ stands for functions from individuals to functions from individuals to functions from individuals to truth-values

Models for the logic are based on hierarchies of typed domains, as defined below:

(2) (TY_2 frames) A TY_2 *frame* is a set $\{D_\alpha \mid \alpha \text{ is a } \mathrm{TY}_2 \text{ type}\}$ such that:

  - $D_e \neq \emptyset$
  - $D_s \neq \emptyset$
  - $D_t = \{0,1\}$
  - $D_{\alpha\beta} \subseteq \{F \mid F : D_\alpha \to D_\beta\}$ for each type $\alpha\beta$

(3) A TY_2 frame is *standard* if $D_{\alpha\beta} = \{F \mid F : D_\alpha \to D_\beta\}$ for each type $\alpha\beta$.

1.1 TY$_2$ syntax

- for each type, we assume a denumerably / countably infinite set of variables of that type
- for each type, we assume a denumerably / countably infinite set of constants
- from these basic expressions, we build up terms inductively with the help of the usual logical connectives, quantification, lambda abstraction, application, and identity

(4) (TY$_2$ terms) For each TY$_2$ type, the set of (TY$_2$) terms of that type is defined as follows:

i. every constant or variable of any type is a term of that type;
ii. if φ and ψ are terms of type t (*formulas*), then $\neg\varphi$ and $(\varphi \wedge \psi)$ are formulas;
iii. if φ is a formula and x is a variable of any type, then $\forall x\, \varphi$ is a formula;
iv. if A is a term of type $\alpha\beta$ and B is a term of type α, then (AB) is a term of type β;
v. if A is a term of type β and x is a variable of type α, then $\lambda x\,(A)$ is a term of type $\alpha\beta$;
vi. if A and B are terms of the same type, then $(A = B)$ is a formula.

The other logical operators have their usual definitions. When parentheses are omitted, association is to the left.

- e.g., instead of writing $(\ldots(AB_1)\ldots B_n)$, we write $AB_1\ldots B_n$

We freely add parentheses where this improves readability. Terms are sometimes subscripted with their types:

- as a metalanguage convention we may write A_α to indicate that A is of type α

These conventions will hold for all subsequent logics.

1.2 TY$_2$ semantics

We can now interpret TY$_2$ terms on TY$_2$ frames – we just need to state how basic expressions (constants and variables) are to be interpreted. This is where interpretation functions and assignments come in.

(5) An *interpretation function* I for a frame $F = \{D_\alpha\}_\alpha$ is a function such that:

- I has the set of all constants as its domain
- $I(c_\alpha) \in D_\alpha$ for each constant c_α of type α

(6) A *standard model* is a tuple $\langle F, I\rangle$ where F is a standard frame and I is an interpretation function for F.

(7) An *assignment* is a function a taking variables as its arguments such that $a(x_\alpha) \in D_\alpha$ for each variable x_α of type α.

(8) If a is an assignment then we write $a[d/x]$ for the assignment a' such that:

- $a'(x) = d$
- $a'(y) = a(y)$ if $x \neq y$

The Tarski truth definition:

(9) Conventions:

- we sometimes write $|A|$ for $|A|^{M,a}$
- we use the von Neumann definition of 0 and 1, so $0 = \emptyset$ and $1 = \{\emptyset\}$

(10) (Tarski truth definition for TY_2) The *value* $|A|^{M,a}$ of a term A on a standard model $M = \langle F, I\rangle$ under an assignment a is defined as follows:

i. $|c| = I(c)$ if c is a constant;
$|x| = a(x)$ if x is a variable;

ii. $|\neg\varphi| = 1 - |\varphi|$;
$|\varphi \wedge \psi| = |\varphi| \cap |\psi|$;

iii. $|\forall x_\alpha\, \varphi|^{M,a} = \bigcap_{d\in D_\alpha} |\varphi|^{M,a[d/x]}$;

iv. $|AB| = |A|(|B|)$;

v. $|\lambda x_\beta\, A|^{M,a} =$ the function F with domain D_β such that $F(d) = |A|^{M,a[d/x]}$ for all $d \in D_\beta$,
i.e., $|\lambda x_\beta\, A|^{M,a} = \big\langle |A|^{M,a[d/x]} : d \in D_\beta \big\rangle$;

vi. $|A = B| = 1$ if $|A| = |B|$,
$\quad = 0$ if $|A| \neq |B|$.

Clauses ii. and iii. are couched completely in terms of the Boolean operations on $\{0, 1\}$ to make partialization easier. These clauses clearly correspond to the usual ones.

The definition of the entailment relation is also put in terms of the natural Boolean algebra on $\{0, 1\}$.

(11) Let Γ and Δ be sets of TY_2 formulas. Γ *s-entails* Δ, $\Gamma \models_s \Delta$, if, for all standard models M and M-assignments a:

$$\bigcap_{\varphi\in\Gamma} |\varphi|^{M,a} \subseteq \bigcup_{\psi\in\Delta} |\psi|^{M,a}$$

(12) A formula φ is *standardly valid* or *s-valid* if $\models_s \varphi$.

This logic behaves classically – the following schemata are s-valid, for example.

(13) (Extensionality) $\forall x\,(Ax = Bx) \rightarrow A = B$

(14) (Universal Instantiation) $\forall x_\alpha\, \varphi \rightarrow [A_\alpha/x]\varphi$

(15) (Lambda Conversion) $\lambda x\,(A)B = [B/x]A$

(16) (Leibniz's Law) $A = B \rightarrow ([A/x]\varphi \rightarrow [B/x]\varphi)$

These laws (except the first) are subject to a substitutability provision:

- in the second schema, A must be free for x in φ
- in the third, B must be free for x in A
- in the fourth, both A and B must be free for x in φ

For more information about TY_2 and its one-sorted and zero-sorted variants (including proof theory), see Henkin (Henkin 1950, 1963) and Gallin (1975).

Since we have decided that indices (elements of D_s) are to be interpreted as world-time pairs, we must ensure that the ground domains D_s of our models will behave in the correct way. We enforce this the help of the non-logical axioms below.

- let $\approx$ and $<$ be two TY_2 constants of type $s(st)$
- a formula $i \approx j$ (we use infix notation here) is to be interpreted as 'i and j have the same world component'
- a formula $i < j$ as 'the time component of i precedes that of j'

The following eight axioms make the domain D_s of indices behave like the Cartesian product of two sets, the second of which is linearly ordered.[1]

(17) AX1 $\forall i\, i \approx i$

AX2 $\forall i \forall j\, (i \approx j \rightarrow j \approx i)$

AX3 $\forall i \forall j \forall k\, (i \approx j \rightarrow (j \approx k \rightarrow i \approx k))$

AX4 $\forall i\, \neg i < i$

AX5 $\forall i \forall j \forall k\, (i < j \rightarrow (j < k \rightarrow i < k))$

AX6 $\forall i \forall j\, (i < j \rightarrow \forall k\, (i < k \vee k < j))$

AX7 $\forall i \forall j \exists k\, (i \approx k \wedge \neg j < k \wedge \neg k < j)$

AX8 $\forall i \forall j\, ((i \approx j \wedge \neg j < i \wedge \neg i < j) \rightarrow i = j)$

Had we worked with a three-sorted logic with two separate sorts for possible worlds and times – instead of our single sort of world-time pairs s, we could have made do with less axioms here:

- the sole purpose of our axioms is to make indices behave as if they were pairs with a strict linear ordering on their second elements – so a set of axioms saying that $<$ is a strict linear order on the domain of times would have sufficed
- the number of basic types can be traded off against the number of axioms here and we have opted for fewer types as this gives an overall simplification of the theory

2 The relational type logic TT$_2$

The logic TY$_2$ can be said to be universal in the sense that all relations and functions over its ground domains, all relations between such functions and relations, and so on, are encodable as objects in its complex domains.

- a ternary relation between individuals can be represented as an object of type $e(e(et))$
- a binary relation between such ternary relations can be coded as an object of type $(e(e(et)))((e(e(et)))t)$
- i.e., we have coded binary relations between ternary relations as functions
  - from: functions from individuals to functions from individuals to functions from individuals to truth values
  - to: functions from functions from individuals to functions from individuals to functions from individuals to truth values to truth values
- we have replaced objects that we have some intuitive grasp on by monsters that we can only reason about in an abstract way

Moreover: if we consider type hierarchies consisting of partial rather than total functions (we need them for θ-roles, for example: not all events have an agent or experiencer role), Schönfinkel's one-one correspondence between multi-argument functions and unary ones breaks down.

For example (adapted from Tichy 1982):

- let a be some object of type e
- consider two partial functions F_1 and F_2, both of type $e(ee)$, defined as follows:
  - $F_1(x) = F_2(x) =$ the identity function, if $x \neq a$
  - $F_1(a)$ is undefined

[1]For a proof of this statement see the proof of Theorem 2 in Chapter 3.

  - $F_2(a)$ is defined as the *ee* function that is undefined for all its arguments

- clearly, $F_1 \neq F_2$
- the function F_2 codes the two-place partial function F such that $F(a, y)$ is undefined and $F(x, y) = y$ if $x \neq a$
- but if F_1 codes anything at all, it must code F too

It seems that it is not a very good idea to put intricate codifications[2] like Schönfinkel's into your logic if they are not absolutely necessary: they complicate the theory.

- if you confine yourself to direct applications of the logic, you may get used to the complications
- but if you are trying to prove things about the logic and generalize it – as it is often needed for natural language semantics, the complications are a hindrance to any real progress

The general idea: we should first give a formulation of type theory that is not based on this identification before we can generalize it to a partial theory of types.

Three options are open:

i. we can consider type hierarchies consisting of both multi-argument functions and multi-argument relations, but we will not need this generality here
ii. we can consider only multi-argument functions; relations can then be coded by their characteristic functions
iii. we can take only relations – and this is the course we will follow, since it is simpler than the first and better suited to our present purposes than the second

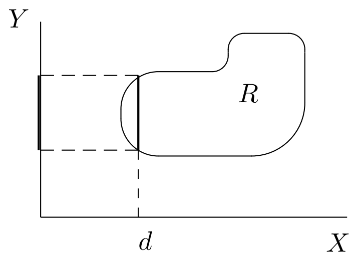
At first glance, it might seem that the last solution is not liberal enough: it is often useful in natural language semantics to take a functional perspective on things.

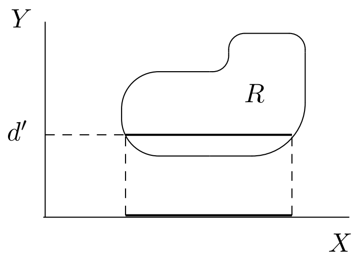
- the *intension* of an expression can fruitfully be seen as a function from indices of evaluation (world-time pairs) to extensions
- functional application seems to be the correct semantic correlate of many / most syntactic constructions

But it is possible to view any relation as a function. And it is possible to do this in at least as many ways as the relation has argument places. The following pictures illustrate this point:

- let R be some binary relation on the reals, or, equivalently, a set of points in the Euclidean plane
- for any point d on the X-axis, we have a corresponding set of points $\{y \mid \langle d, y\rangle \in R\}$ on the Y-axis – see the left picture
- for any point d' on the Y-axis, we have a corresponding set of points $\{x \mid \langle x, d'\rangle \in R\}$ on the X-axis – see the right picture
- thus: there are two natural ways to see R as a function from the reals to the power set of the reals
- for binary relations, e.g., *kiss*, *devour* etc., we choose one particular way for syntactic purposes – but both 'schönfinkelizations' / ways of 'currying' are equally plausible from a semantic point of view

[2]The question whether the Schönfinkel encoding is intricate or not is of course a matter of taste and opinion. Note however that a double recursion is needed if one wants to define the correspondence in any precise way. See the definition of the functions S_α in the next section.





The procedure is entirely general:

(18) (Slice Functions) Let R be an n-ary relation ($n > 0$) and let $0 < k \leq n$. Define the *k-th slice function* of R by:

$$F^k_R(d) = \{\langle d_1, \ldots, d_{k-1}, d_{k+1}, \ldots, d_n\rangle \mid \langle d_1, \ldots, d_{k-1}, d, d_{k+1}, \ldots, d_n\rangle \in R\}.$$

- $F^k_R(d)$ is the $n-1$-ary relation that is obtained from R by fixing its k-th argument place by d (we will often want to view relations as functions in this way)
- given that $\langle a\rangle = a$, $\langle\,\rangle = \emptyset$, $\emptyset = 0$ and $\{\emptyset\} = 1$:
  if R is a one-place relation, then F^1_R is its characteristic function
- for example:
  - let *love* be a ternary relation, *love* xyi meaning 'y loves x at index i'
  - this is a relation-in-intension; relations-in-intension are normally thought of as functions from possible worlds to extensions – F^3_{love} is this function for *love*
  - it is also natural to view the relation as the function that, when applied to an entity 'Mary' gives the property 'y loves Mary at index i' – this is the function F^1_{love}
- thus: we can shift between a functional and a relational perspective without making use of Schönfinkel's identification

2.1 TT$_2$ types and frames

A relational formulation of higher-order logic was given in Orey (1959) (see also Gallin 1975 and and Benthem and Doets 1983). The following two definitions give a two-sorted version of Orey's type hierarchies:

(19) (TT$_2$ types) The set of types is the smallest set of strings such that:

i. e and s are types,

ii. if $\alpha_1, \ldots, \alpha_n$ are types ($n \geq 0$), then $\langle\alpha_1 \ldots \alpha_n\rangle$ is a type.

(20) (TT$_2$ frames) A *frame* is a set of sets $\{D_\alpha \mid \alpha \text{ is a type}\}$ such that $D_e \neq \emptyset$, $D_s \neq \emptyset$ and

$$D_{\langle\alpha_1\ldots\alpha_n\rangle} \subseteq Pow(D_{\alpha_1} \times \ldots \times D_{\alpha_n})$$

for all types $\alpha_1, \ldots, \alpha_n$.

(21) A frame is *standard* if

$$D_{\langle\alpha_1\ldots\alpha_n\rangle} = Pow(D_{\alpha_1} \times \ldots \times D_{\alpha_n})$$

for all types $\alpha_1, \ldots, \alpha_n$.

- **note**: the angled brackets are crucial and cannot be omitted – we do not want to equate the type $\langle e\rangle$ with the type e, although we usually do equate the ordered 1-tuple $\langle a\rangle$ with a

- domains D_e and D_s are thought to consist of possible individuals and world-time pairs respectively
- domains $D_{\langle\alpha_1\ldots\alpha_n\rangle}$ consist of all n-ary relations having D_{α_i} as their i-th domain
- the string $\langle\,\rangle$ is a type and $D_{\langle\,\rangle} = Pow(\{\emptyset\}) = \{0,1\}$ – i.e., the set of *truth-values*

Orey used his relational frames to interpret the formulas of higher-order predicate logic on.

- these formulas have a syntax that is essentially that of ordinary predicate logic – except that quantification over objects of arbitrary type is allowed
- there is no lambda-abstraction and the syntax allows only one type of complex expressions: type $\langle\,\rangle$, the type of formulas
- Montague Grammar assigns many different types to linguistic phrases – so, higher-order predicate logic as it stands does not fit our purposes
- but the syntax of ordinary functional type logic does satisfy our needs
- the plan: keep Church's syntax and attach Orey's models to it

2.2 TT$_2$ syntax

We need a slight reformulation of the syntax – more precisely, of the typing of terms.

- for each type, assume a denumerably infinite set of variables of that type
- for each type, assume a countably infinite set of constants

(22) (TT$_2$ terms) For each type α, the set of *terms* of type α is defined as follows:

i. every constant or variable of any type is a term of that type;
ii. if φ and ψ are terms of type $\langle\,\rangle$ (*formulas*), then $\neg\varphi$ and $(\varphi \wedge \psi)$ are formulas;
iii. if φ is a formula (type $\langle\,\rangle$) and x is a variable of any type, then $\forall x\,\varphi$ is a formula;
iv. if A is a term of type $\langle\beta\alpha_1\ldots\alpha_n\rangle$ and B is a term of type β, then (AB) is a term of type $\langle\alpha_1\ldots\alpha_n\rangle$;
v. if A is a term of type $\langle\alpha_1\ldots\alpha_n\rangle$ and x is a variable of type β, then $\lambda x\,(A)$ is a term of type $\langle\beta\alpha_1\ldots\alpha_n\rangle$;
vi. if A and B are terms of the same type, then $(A = B)$ is a formula (type $\langle\,\rangle$).

2.3 TT$_2$ semantics

(23) *Standard models* are tuples $\langle F, I\rangle$, consisting of a frame $F = \{D_\alpha\}_\alpha$ and an *interpretation function* I such that:

- I has the set of constants as its domain
- $I(c) \in D_\alpha$ for each constant c of type a

(24) An *assignment* is a function a taking variables as its arguments such that $a(x_\alpha) \in D_\alpha$ for each variable x_α of type α.

(25) If a is an assignment then we write $a[d/x]$ for the assignment a' such that:

- $a'(x) = d$
- $a'(y) = a(y)$ if $x \neq y$

We use the slice functions defined above to evaluate TT$_2$ terms on these relational standard models:

- we simply let the value of a term AB be the result of applying the first slice function of the value of A to the value of B
- we evaluate terms of the form $\lambda x.\, A$ by an inverse procedure

(26) (Tarski truth definition for TT$_2$) The *value* $\|A\|^{M,a}$ of a term A on a model M under an assignment a is defined in the following way (we sometimes write $\|A\|$ for $\|A\|^{M,a}$ to improve readability):

i. $\|c\| = I(c)$ if c is a constant;
$\|x\| = a(x)$ if x is a variable;

ii. $\|\neg\varphi\| = 1 - \|\varphi\|$;
$\|\varphi \wedge \psi\| = \|\varphi\| \cap \|\psi\|$;

iii. $\|\forall x_\alpha\, \varphi\|^{M,a} = \bigcap_{d \in D_\alpha} \|\varphi\|^{M,a[d/x]}$;

iv. $\|AB\| = F^1_{\|A\|}(\|B\|)$;

v. $\|\lambda x_\beta\, A\|^{M,a} =$ the R such that $F^1_R(d) = \|A\|^{M,a[d/x]}$ for all $d \in D_\beta$,
i.e., the R such that $F^1_R = \langle \|A\|^{M,a[d/x]} : d \in D_\beta \rangle$;

vi. $\|A = B\| = 1$ if $\|A\| = \|B\|$;
$= 0$ if $\|A\| \neq \|B\|$.

- clauses i, ii, iii and vi are completely analogous to the corresponding TY$_2$ clauses
- only clauses iv and v differ

The following identities hold:

$$\begin{aligned}
\|AB\| &= \{\langle d_1, \ldots, d_n\rangle \mid \langle \|B\|, d_1, \ldots, d_n\rangle \in \|A\|\} \\
\|\lambda x_\beta\, A\|^{M,a} &= \{\langle d, d_1, \ldots, d_n\rangle \mid d \in D_\beta \text{ and} \\
&\qquad \langle d_1, \ldots, d_n\rangle \in \|A\|^{M,a[d/x]}\}.
\end{aligned}$$

Given the Boolean character of our relational domains, it is possible to define the notion of logical consequence for terms of arbitrary relational type, not only for formulas.

(27) (Entailment in TT$_2$) Let Γ and Δ be sets of terms of some type $\alpha = \langle \alpha_1 \ldots \alpha_n \rangle$. Γ is said to *s-entail* Δ, $\Gamma \models_s \Delta$, if

$$\bigcap_{A \in \Gamma} \|A\|^{M,a} \subseteq \bigcup_{B \in \Delta} \|B\|^{M,a}$$

for all standard models M and M-assignments a.

3 The logics TY$_2$ and TT$_2$ compared

Comparing our relational frames with the more usual functional ones:

- one one hand: since every function is a relation, it should be clear that all objects occurring in some functional standard frame occur in the relational standard frame based on the same ground domains D_e and D_s as well
- on the other hand: we can code relations as functions by the Schönfinkel codification

To give a formal account of this encoding, we first we need to establish a correspondence between relational types and functional ones.

(28) Define the function Σ (Σ is for Schönfinkel) taking TT$_2$ types to TY$_2$ types by the following double recursion:

I $\Sigma(e) = e$, $\Sigma(s) = s$

II i. $\Sigma(\langle\,\rangle) = t$

ii. $\Sigma(\langle\alpha_1 \ldots \alpha_n\rangle) = (\Sigma(\alpha_1)\Sigma(\langle\alpha_2 \ldots \alpha_n\rangle))$ if $n \geq 1$.

For example:

- $\Sigma(\langle e\rangle) = et$
- $\Sigma(\langle\langle e\rangle\rangle) = (et)t$
- $\Sigma(\langle ee\rangle) = e(et)$
- $\Sigma(\langle\langle se\rangle\langle se\rangle\rangle) = (s(et))((s(et))t)$

In general, if α is the type of some relation, then $\Sigma(\alpha)$ is the type of the unary function that codes this relation in functional type theory.

The arguments of Σ, i.e., the TT_2 types, tend to have less length than the corresponding values, i.e., the TY_2 types.

(29) We call any TY_2 type that is a value of Σ *quasi-relational*.

(30) A TY_2 type is a value of Σ, i.e., is a quasi-relational type, iff no occurrence of e or s immediately precedes a right bracket in it.

- this characterization presupposes the 'official' notation for functional types, with outer brackets in place
- for example, in order to avoid clutter, we usually write se for (se), but this type is not quasi-relational

Next step: provide a full definition of the Schönfinkel encoding function.

- this function is simple if only relations over individuals are considered
- in the higher-order case, where relations can take relations as arguments, which in their turn can again take relations as arguments and so on, the identification is somewhat less transparent

(31) (The Schönfinkel Encoding) Let $F = \{D_\alpha \mid \alpha \text{ is a TT}_2 \text{ type type}\}$ be a standard TT_2 frame and let $F' = \{D'_\alpha \mid \alpha \text{ is a TY}_2 \text{ type}\}$ be the TY_2 standard frame such that $D_e = D'_e$ and $D_s = D'_s$.
For each TT_2 type α, we define a function $S_\alpha : D_\alpha \to D'_{\Sigma(\alpha)}$ by the following double recursion:

I $S_e(d) = d$, if $d \in D_e$; $S_s(d) = d$, if $d \in D_s$;

II i. $S_{\langle\,\rangle}(d) = d$, if $d \in D_{\langle\,\rangle}$;

ii. if $n \geq 1$, $\alpha = \langle\alpha_1 \ldots \alpha_n\rangle$ and $R \in D_\alpha$, then $S_\alpha(R)$ is the function G of type $(\Sigma(\alpha_1)\Sigma(\langle\alpha_2 \ldots \alpha_n\rangle))$ such that:
$G(f) = S_{\langle\alpha_2 \ldots \alpha_n\rangle}(F^1_R(S^{-1}_{\alpha_1}(f)))$ for each $f \in D'_{\Sigma(\alpha_1)}$.

- the functions S_α are bijections, so the definition is correct

This definition shows us two things:

- we *can* code multi-argument relations as unary functions:
$\langle d_1, \ldots, d_n\rangle \in R$ iff $S(R)(S(d_1)) \ldots (S(d_n)) = 1$, for all relations R (of any type)
- we *shouldn't* code multi-argument relations as unary functions:
the functions S_α tend to increase complexity rather dramatically; this doubly recursive encoding is just a needless complication.

3.1 An example of redundancy attributable to the Schönfinkel encoding

Natural language and, or and not can be used with expressions of almost all linguistic categories – so, type domains should have a Boolean structure. This has been argued for by a variety of authors, beginning with Von Stechow (1974) (see also Keenan and Faltz (1978)).

Orey's relational standard frames have a Boolean structure on all their (non-basic) domains, since they are power sets:

- the rule for the interpretation of natural language conjunction, disjunction and negation is very simple: they are to be treated as $\cap$, $\cup$ and $-$ (complementation within a typed domain) respectively
- entailment between expressions of the same category is to be treated as inclusion

But: the relevant Boolean operations are not as easily available in a functional type theory. Therefore Gazdar (1980) (see also the work of Partee & Rooth) gives some pointwise recursive definitions.

First, we must characterize a certain subclass of the TY_2 types, the so-called *conjoinable* ones:

(32) (Conjoinable TY_2 types)

i. t is conjoinable;

ii. if β is conjoinable, then $\alpha\beta$ is conjoinable.

- while not all conjoinable TY_2 types are quasi-relational, the two classes of types are closely related: a TY_2 type is quasi-relational if and only if all its subtypes are either basic or conjoinable

The definition of generalized conjunction in functional type theory is as follows:

(33) (Generalized Conjunction)

i. $a \sqcap b := a \cap b$, if $a, b \in \{0, 1\}$ (i.e., a, b are of type t);

ii. if F_1 and F_2 are functions of some conjoinable TY_2 type $\alpha\beta$, $(F_1 \sqcap F_2)(z) := \lambda z_\alpha\, F_1(z) \sqcap F_2(z)$.

- similar definitions can be given for generalized disjunction, complementation and inclusion (see Groenendijk and Stokhof (1984) for the last operation)
- but the need for these definitions is an artifact of Schönfinkel's Trick – they enable us to treat generalized coordination by reversing Schönfinkelizations
- more precisely: for any R_1, R_2 of relational type, $S(R_1 \cap R_2) = S(R_1) \sqcap S(R_2)$
- as soon as we get rid of the Trick, the need for its reversals (i.e., these pointwise definitions) vanishes too

We can use the formal characterization of the Schönfinkel Encoding to prove an equivalence between our two logics.

- first, we stipulate that the constants (variables) of any TT_2 type α are identical to the constants (variables) of TY_2 type $\Sigma(\alpha)$
- given this stipulation, it is easily seen that all TT_2 terms of any type α are TY_2 terms of type $\Sigma(\alpha)$ (and vice versa)
- so our new syntax is just a part of the TY_2 syntax
- not all TY_2 terms are TT_2 terms by this identification since Σ is not onto, but the TY_2 terms that belong to TT_2 are exactly those whose subterms are all of a quasi-relational type

The following theorem (proved is in the Appendix) says that both logics give the same entailment relation on TT_2 sentences. That is: we can have the nice models of the relational theory, but we do not have to give up the classical entailment relation. In particular, laws such as Extensionality, Lambda Conversion, Universal Instantiation and Leibniz's Law continue to hold.

(34) Let Γ and Δ be sets of TT_2 sentences. Then $\Gamma \models_s \Delta$ in TT_2 iff $\Gamma \models_s \Delta$ in TY_2.

References

Barwise, J. and R. Cooper: 1981, 'Generalized Quantifiers and Natural Language', *Linguistics and Philosophy* **4**, 159–219.

Benthem, J.F.A.K. van and K. Doets: 1983, 'Higher-Order Logic', in D. Gabbay and F. Guenthner (eds.), *Handbook of Philosophical Logic*, Vol. I, 275–329. Reidel, Dordrecht.

Church, A.: 1940, 'A Formulation of the Simple Theory of Types', *Journal of Symbolic Logic* **5**, 56–68.

Gallin, D.: 1975, *Intensional and Higher-Order Modal Logic.* North-Holland, Amsterdam.

Gazdar, G.: 1980, 'A Cross-Categorial Semantics for Coordination', *Linguistics and Philosophy* **3**, 407–409.

Groenendijk, J. and M. Stokhof: 1984, *Studies on the Semantics of Questions and the Pragmatics of Answers*, Doctoral Dissertation, University of Amsterdam.

Henkin, L.: 1950, 'Completeness in the Theory of Types', *Journal of Symbolic Logic* **15**, 81–91.

Henkin, L.: 1963, 'A Theory of Propositional Types', *Fundamenta Mathematicae* **52**, 323–344.

Keenan, E. and L. Faltz: 1978, 'Logical Types for Natural Language'. UCLA Occasional Papers in Linguistics, 3.

Orey, S.: 1959, 'Model Theory for the Higher Order Predicate Calculus', *Transactions of the American Mathematical Society* **92**, 72–84.

Russell, B.: 1908, 'Mathematical Logic as Based on the Theory of Types', *American Journal of Mathematics* **30**, 222–262.

Tichy, P.: 1982, 'Foundations of Partial Type Theory', *Reports on Mathematical Logic* **14**, 59–72.

Von Stechow, A.: 1974, 'ϵ-λ kontextfreie Sprachen: Ein Beitrag zu einer natürlichen formalen Semantik', *Linguistische Berichte* **34**, 1–33.