Rendering Natural Language of Mathematical Texts into Formal Language

Roussanka Loukanova

Institute of Mathematics and Informatics (IMI)
Bulgarian Academy of Sciences (BAS), Sofia, Bulgaria

NatFoM: Workshop on Natural Formal Mathematics, Cambridge, 6 September 14:00–18:00

Joint WG4-WG5 meeting, Cambridge, UK, 6-8 September 2023
https://europroofnet.github.io/cambridge-2023

Outline

1. A Glimpse of Approaches to Formal and Computational Grammar
   - Overview of Approaches to Computational Semantics
2. Syntax and Denotational Semantics of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$
   - Syntax of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$
   - Denotational Semantics of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ / $\mathrm{L}^{\lambda}_{rar}$
3. Reduction Calculi, Canonical Forms, and Algorithmic Semantica
   - $\gamma*$-Reduction
   - Canonical Forms and Algorithmic Semantics
   - Algorithmic Equivalence
   - Expressiveness of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ / L^{λ}_{r}
4. Parametric Algorithmic Patterns
   - Pure Quantifiers
   - Generalised Quantifiers, Algorithmic Patterns, Ambiguity, Underspecification
   - Definite Descriptors with Determiner "the"
   - Conjuncts and Coordination
5. Computational Syntax-Semantics of NL via $\mathrm{L}^{\lambda}_{\mathrm{ar}}$

Approaches to formal and computational syntax of natural language (NL)

All of the following approaches are at least partly active

CFGs, Phrase Structure Grammars (PSG): initiated by Chomsky 1950s

Transformational Grammars: initiated by Chomsky 1955, 1957, with versions to the present

Generative Semantics: 1967-74 Lakoff, McCawley, Postal, Ross

Government and Binding Theory (GBT): initiated by Chomsky 1981

Principles and Parameters initiated by Chomsky 1981 with GBT

Minimalist Program initiated by Chomsky 1995 (major work)

Constraint-Based, Lexicalist Approaches

- *GPSG:* Gazdar et al. 1979-87 to the present
- *LFG:* 1979 to the present
- *HPSG:* 1984 to the present

Categorial Grammars Ajdukiewicz 1935 to the present

Dependency Grammar (DG): active

Grammatical Framework (GF) Multi-Lingual, Chalmers, 1998, Aarne Ranta (25 years on, in Mar 2023) (open development)

...

- *Categorial Grammars:* Ajdukiewicz 1935 — formal logic for syntax for NL to the present, with initiations for syntax-semantics
- *Type-Theoretical Grammars* in many varieties
- *Montague Grammars:* started by Montague 1970 to the present
- Situation Theory and Situation Semantics, Jon Barwise 1980ies
  Inspired partiality in computational syntax of LFG and HPSG;
  Since start HPSG approaches, 1984, have been using Situation Semantics in syntax-semantics interfaces;
- *Minimal Recursion Semantics* in HPSG since 2000-2002
  MRS is a technique as a form of Situation Semantics with major characteristics of Moschovakis recursion
- Moschovakis [12] Formal Language of full recursion, untyped;
  Typed acyclic recursion, introduced by Moschovakis [13] (2006)
- Algorithmic Dependent-Type Theory of Situated Information (DTTSitInfo): situated data including context assessments (open)
- Other Approaches to Computational Semantics many combinations and variants of FOL, e.g., Prolog, Definite Clause Grammars, etc.

Algorithms for computing denotations of terms

Algorithmic syntax-semantics of L^{λ}_{ar} (L^{λ}_{r}) and Natural Language

$$\underbrace{\text{Syntax of } L^{\lambda}_{ar}\ (L^{\lambda}_{r}) \Longrightarrow \text{Algorithms for Computations} \Longrightarrow \text{Denotations}}_{\text{Semantics of } L^{\lambda}_{ar}(L^{\lambda}_{r})} \tag{1}$$

$$\underbrace{\text{Computational Syntax of NL}}_{\text{Computational Grammar}} \xrightarrow{\text{render}} L^{\lambda}_{ar} \tag{2}$$

$$\underbrace{\text{Computational Syntax of NL} \xrightarrow{\text{render}} L^{\lambda}_{ar}}_{\text{Computational Grammar: Syntax-Semantics Interface}} \tag{3}$$

Development of Type-Theory of (Acyclic) Algorithms, L_r^{λ} (L_{ar}^{λ})

Placement of L_{ar}^{λ} in a class of type theories

$$\text{Montague IL} \subsetneq \text{Gallin TY}_2 \subsetneq \text{Moschovakis } L_{ar}^{\lambda} \subsetneq \text{Moschovakis } L_r^{\lambda} \tag{4}$$

- Type-Theory of (Acyclic) Algorithms, L_r^{λ} (L_{ar}^{λ}): provides:
  - a math notion of algorithm
  - Computational Semantics of formal and natural languages
- L_{ar}^{λ} / L_r^{λ} is type theory of algorithms with acyclic / full recursion:
  - Introduced by Moschovakis [13] (2006),
  - Math development by motivations from NL, Loukanova [8, 9] (2019) and previously
- In the works presented here, I extend L_{ar}^{λ} / L_r^{λ} by incorporating
  - logic operators, by logic constants of suitable types
  - pure, logic quantifiers
  - extended reduction calculus of L_{ar}^{λ} / L_r^{λ}
  - demonstrate (there is a math proof) that L_{ar}^{λ} / L_r^{λ} essentially extend classic λ-calculus,

    incl., for logic operators and pure quantifiers

Syntax of Type Theory of Algorithms (TTA): Types, Vocabulary

- Gallin Types (1975)

$$\tau ::= \mathsf{e} \mid \mathsf{t} \mid \mathsf{s} \mid (\tau \to \tau) \qquad \text{(Types)}$$

- Abbreviations

$$\widetilde{\sigma} \equiv (\mathsf{s} \to \sigma), \quad \text{for state-dependent objects of type } \widetilde{\sigma} \qquad (5a)$$

$$\widetilde{\mathsf{e}} \equiv (\mathsf{s} \to \mathsf{e}), \quad \text{for state-dependent entities} \qquad (5b)$$

$$\widetilde{\mathsf{t}} \equiv (\mathsf{s} \to \mathsf{t}), \quad \text{for state-dependent truth values} \qquad (5c)$$

- Typed Vocabulary, for all $\sigma \in$ Types

$$K_\sigma = \mathsf{Consts}_\sigma = \{\mathsf{c}_0^\sigma, \mathsf{c}_1^\sigma, \dots\} \qquad (6a)$$

$$\wedge, \vee, \to \ \in \mathsf{Consts}_{(\tau\to(\tau\to\tau))}, \ \tau \in \{\mathsf{t}, \widetilde{\mathsf{t}}\} \quad \text{(logical constants)} \qquad (6b)$$

$$\neg \in \mathsf{Consts}_{(\tau\to\tau)}, \ \tau \in \{\mathsf{t}, \widetilde{\mathsf{t}}\} \quad \text{(logical constant for negation)} \qquad (6c)$$

$$\mathsf{PureV}_\sigma = \{v_0^\sigma, v_1^\sigma, \dots\} \quad \text{(pure variables)} \qquad (6d)$$

$$\mathsf{RecV}_\sigma = \mathsf{MemoryV}_\sigma = \{p_0^\sigma, p_1^\sigma, \dots\} \quad \text{(recursion variables)} \qquad (6e)$$

$$\mathsf{PureV}_\sigma \cap \mathsf{RecV}_\sigma = \varnothing, \qquad \mathsf{Vars}_\sigma = \mathsf{PureV}_\sigma \cup \mathsf{RecV}_\sigma \qquad (6f)$$

$$\begin{aligned}
\mathsf{A} :\equiv\ & \mathsf{c}^{\sigma} : \sigma \mid X^{\sigma} : \sigma \mid \mathsf{B}^{(\sigma\to\tau)}(\mathsf{C}^{\sigma}) : \tau \mid \lambda(v^{\sigma})\,(\mathsf{B}^{\tau}) : (\sigma\to\tau) && \text{(7a)}\\
& \mid \mathsf{A}_0^{\sigma_0} \text{ where } \{\, p_1^{\sigma_1} := \mathsf{A}_1^{\sigma_1}, \ldots, p_n^{\sigma_n} := \mathsf{A}_n^{\sigma_n} \,\} : \sigma_0 && \text{(7b)}\\
& \mid \wedge\,(A_2^{\tau})(A_1^{\tau}) : \tau \mid \vee\,(A_2^{\tau})(A_1^{\tau}) : \tau \mid \to(A_2^{\tau})(A_1^{\tau}) : \tau && \text{(7c)}\\
& \mid \neg(B^{\tau}) : \tau && \text{(7d)}\\
& \mid \forall(v^{\sigma})(B^{\tau}) : \tau \mid \exists(v^{\sigma})(B^{\tau}) : \tau \qquad \text{(pure quantifiers)} && \text{(7e)}\\
& \mid \mathsf{A}_0^{\sigma_0} \text{ such that } \{\, \mathsf{C}_1^{\tau_1}, \ldots, \mathsf{C}_m^{\tau_m} \,\} : \sigma_0' \qquad \text{(restrictor operator)} && \text{(7f)}
\end{aligned}$$

- $\mathsf{c}^{\tau} \in \mathsf{Consts}_{\tau}$, $X^{\tau} \in \mathsf{PureV}_{\tau} \cup \mathsf{RecV}_{\tau}$, $v^{\sigma} \in \mathsf{PureV}_{\sigma}$
- $\mathsf{B}, \mathsf{C} \in \mathsf{Terms}$, $p_i^{\sigma_i} \in \mathsf{RecV}_{\sigma_i}$, $A_i^{\sigma_i} \in \mathsf{Terms}_{\sigma_i}$, $\mathsf{C}_j^{\tau_j} \in \mathsf{Terms}_{\tau_j}$
- In (7c)–(7e), (7f): $\tau, \tau_j \in \{\, \mathsf{t}, \widetilde{\mathsf{t}} \,\}$, $\widetilde{\mathsf{t}} \equiv (\mathsf{s} \to \mathsf{t})$ (for propositions)
- Acyclicity Constraint (AC), for $\mathrm{L}^{\lambda}_{\mathrm{ar}}$; without it, L^{λ}_{r} with full recursion

$$\{\, p_1^{\sigma_1} := A_1^{\sigma_1}, \ldots, p_n^{\sigma_n} := A_n^{\sigma_n} \,\} \quad (n \geq 0) \text{ is acyclic iff} \tag{8a}$$

for some function $\mathsf{rank}\colon \{p_1, \ldots, p_n\} \to \mathbb{N}$

$$\text{if} \quad p_j \in \mathsf{FreeV}(A_i) \ (p_j \text{ occurs freely in } A_i), \tag{8b}$$

$$\text{then} \quad \mathsf{rank}(p_i) > \mathsf{rank}(p_j)$$

Types of Restrictor Terms

In the restrictor term (7f) / (9),

$$A_0^{\sigma_0} \text{ such that } \{ C_1^{\tau_1}, \dots, C_n^{\tau_n} \} : \sigma'_0 \tag{9}$$

for each $i = 1, \dots, n$:

- $\tau_i \equiv \mathsf{t}$ (state independent truth values), or
- $\tau_i \equiv \widetilde{\mathsf{t}} \equiv (\mathsf{s} \to \mathsf{t})$ (state dependent truth values)

$$\sigma'_0 \equiv \begin{cases} \sigma_0, & \text{if } \tau_i \equiv \mathsf{t}, \text{ for all } i \in \{ 1, \dots, n \} \quad \text{(10a)} \\ \sigma_0 \equiv (\mathsf{s} \to \sigma), & \text{if } \tau_i \equiv \widetilde{\mathsf{t}}, \text{ for some } i \in \{ 1, \dots, n \}, \text{ and} \quad \text{(10b)} \\ & \quad \text{for some } \sigma \in \mathsf{Types},\ \sigma_0 \equiv (\mathsf{s} \to \sigma) \\ \widetilde{\sigma_0} \equiv (\mathsf{s} \to \sigma_0), & \text{if } \tau_i \equiv \widetilde{\mathsf{t}}, \text{ for some } i \in \{ 1, \dots, n \}, \text{ and} \quad \text{(10c)} \\ & \quad \text{there is no } \sigma, \text{ s.th. } \sigma_0 \equiv (\mathsf{s} \to \sigma) \end{cases}$$

Denotational Semantics of L^{λ}_{ar} / L^{λ}_{rar}

A standard semantic structure is a tuple $\mathfrak{A}(\mathsf{Consts}) = \langle\, \mathbb{T}, \mathcal{I}\,\rangle$ that satisfies the following conditions:

- $\mathbb{T} = \{\mathbb{T}_\sigma \mid \sigma \in \mathsf{Types}\}$ is a frame of typed objects
  $\{\,0, 1, er\,\} \subseteq \mathbb{T}_{\mathsf{t}} \subseteq \mathbb{T}_{\mathsf{e}}$ $\;(er_{\mathsf{t}} \equiv er_{\mathsf{e}} \equiv er \equiv error)$
  $\mathbb{T}_{\mathsf{s}} \neq \varnothing$ (the domain of *states*)
  $\mathbb{T}_{(\tau_1 \to \tau_2)} = (\mathbb{T}_{\tau_1} \to \mathbb{T}_{\tau_2}) = \{\, f \mid f \colon \mathbb{T}_{\tau_1} \to \mathbb{T}_{\tau_2}\,\}$ (standard str.)
  $er_\sigma \in \mathbb{T}_\sigma$, for every $\sigma \in \mathsf{Types}$ (designated typed errors)
- $\mathcal{I}\colon\ \mathsf{Consts} \longrightarrow \cup\,\mathbb{T}$ is a typed *interpretation function*:
  $\mathcal{I}(\mathsf{c}) \in \mathbb{T}_\sigma$, for every $\mathsf{c} \in \mathsf{Consts}_\sigma$
- $\mathfrak{A}$ is associated with the set of the typed variable valuations G:

$$G = \{g \mid g\colon\ \mathsf{PureV} \cup \mathsf{RecV} \longrightarrow \bigcup \mathbb{T} \quad \text{and, for every } X \in \mathsf{Vars}_\sigma,\ \ g(X) \in \mathbb{T}_\sigma\} \tag{11}$$

The Denotation Function of L^{λ}_{ar} / L^{λ}_{ar} (to be continued)

- Let's assume a given semantic structure $\mathfrak{A}$, and write $\mathrm{den} \equiv \mathrm{den}^{\mathfrak{A}}$
- There is a unique function, called the *denotation function*:
  $\mathrm{den}^{\mathfrak{A}} \colon \mathrm{Terms} \longrightarrow \{ f \mid f \colon G \longrightarrow \cup \mathbb{T} \}$
  defined by recursion on the structure of the terms

(D1)
1. $\mathrm{den}(X)(g) = g(x)$, for every $X \in \mathrm{Vars}$
2. $\mathrm{den}(\mathsf{c})(g) = \mathcal{I}(\mathsf{c})$, for every $\mathsf{c} \in \mathrm{Consts}$

(D2) $\mathrm{den}(A(B))(g) = \mathrm{den}(A)(g)(\mathrm{den}(B)(g))$

(D3) $\mathrm{den}(\lambda x(B))(g)\big(a\big) = \mathrm{den}(B)(g\{x := a\})$, for every $a \in \mathbb{T}_{\tau}$

The Denotation of the Recursion Terms (continuation) (to be continued)

(D4) $\mathsf{den}(A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\})(g) =$
$\mathsf{den}(A_0)(g\{p_1 := \overline{p}_1, \ldots, p_n := \overline{p}_n\})$

where $\overline{p}_i \in \mathbb{T}_{\tau_i}$ are defined by recursion on $\mathsf{rank}(p_i)$:

$$\overline{p_i} = \mathsf{den}(A_i)(g\{p_{k_1} := \overline{p}_{k_1}, \ldots, p_{k_m} := \overline{p}_{k_m}\})$$

given that $p_{k_1}, \ldots, p_{k_m}$ are all of the recursion variables $p_j \in \{p_1, \ldots, p_n\}$, s.t. $\mathsf{rank}(p_j) < \mathsf{rank}(p_i)$.

Intuitively:

- $\mathsf{den}(A_1)(g), \ldots, \mathsf{den}(A_n)(g)$ are computed recursively, by $\mathsf{rank}(p_i)$, and stored in p_i, $1 \leq i \leq n$
- the denotation $\mathsf{den}(A_0)(g)$ may depend on the values stored in $p_1, \ldots, p_n$

(D5) (for the constants of the logic operators) ...

The Denotation of the Logic-Quantifiers Terms (continuation) (to be continued)

(D6b) Simplified version, without considering the erroneous cases of er

The denotation of the state-dependent, pure existential quantifier, for $\tau = \widetilde{\mathsf{t}}$, $\mathrm{den}^{\mathfrak{A}}\big(\exists(v^{\sigma})(B^{\tau})\big)(g) : \mathbb{T}_{\mathsf{s}} \to \mathbb{T}_{\mathsf{t}}$ is such that:

for every state $s \in \mathbb{T}_{\mathsf{s}}$: (12a)

$$\big[\mathrm{den}^{\mathfrak{A}}\big(\exists(v^{\sigma})(B^{\tau})\big)(g)\big](s) = 1 \ \text{(true in } s\text{)} \tag{12b}$$

iff there is $a \in \mathbb{T}_{\sigma}$, in the semantic domain $\mathbb{T}_{\sigma}$, such that:

$$\big[\mathrm{den}^{\mathfrak{A}}\big(B^{\tau}\big)(g\{\, v := a \,\})\big](s) = 1 \tag{12c}$$

The Denotation Function for the Restrictor Terms (continuation) (to be continued)

(D7) For every $g \in G$, and every state $s \in \mathbb{T}_{\mathsf{s}}$:

Case 1: for all $i \in \{1, \dots, n\}$, $C_i \in \mathsf{Terms}_{\mathsf{t}}$ (independent on states)

For every $g \in G$:

$$\mathsf{den}\left(A_0^{\sigma_0} \text{ s.t. } \{\, \overrightarrow{C} \,\}\right)(g) = \begin{cases} \mathsf{den}(A_0)(g), & \text{if, for all } i \in \{1, \dots, n\}, \\ & \quad \mathsf{den}(C_i)(g) = 1 \\ & \\ er_{\sigma_0} & \text{if, for some } i \in \{1, \dots, n\}, \\ & \quad \mathsf{den}(C_i)(g) = 0 \text{ or} \\ & \quad \mathsf{den}(C_i)(g) = er \end{cases} \tag{13}$$

Case 2: for some $i \in \{1, \ldots, n\}$, $C_i : \tilde{\mathsf{t}}$

$$
\mathsf{den}\left(A_0^{\sigma_0} \text{ s.t. } \{ \overrightarrow{C} \}\right)(g)(s) \tag{14}
$$
$$
= \begin{cases} \mathsf{den}(A_0)(g)(s), & \text{if } \mathsf{den}(C_i)(g) = 1, \text{ for all } i \text{ s.th. } C_i : \mathsf{t}, \text{ and} \\ & \quad \mathsf{den}(C_i)(g)(s) = 1, \text{ for all } i \text{ s.th. } C_i : \tilde{\mathsf{t}}, \text{ and} \\ & \quad \sigma_0 \equiv (\mathsf{s} \to \sigma) \\ \mathsf{den}(A_0)(g), & \text{if } \mathsf{den}(C_i)(g) = 1, \text{ for all } i \text{ s.th. } C_i : \mathsf{t}, \text{ and} \\ & \quad \mathsf{den}(C_i)(g)(s) = 1, \text{ for all } i \text{ s.th. } C_i : \tilde{\mathsf{t}}, \text{ and} \\ & \quad \sigma_0 \not\equiv (\mathsf{s} \to \sigma), \quad \text{for all } \sigma \in \mathsf{Types} \\ er_{\sigma_0'}, & \text{otherwise} \end{cases}
$$

- $A \in$ Terms is explicit iff the operator where does not occur in A
- $A \in$ Terms is a *λ-calculus term* iff it is explicit and no recursion variable occurs in it

Definition (Immediate and Proper Terms)

- The set ImT of immediate terms is defined by recursion (15)

$$T :\equiv V \mid p(v_1)\dots(v_m) \mid \lambda(u_1)\dots\lambda(u_n)p(v_1)\dots(v_m) \tag{15}$$

for $V \in$ Vars, $p \in$ RecV, $u_i, v_j, \in$ PureV,
$i = 1, \dots, n,\ \ j = 1, \dots, m\ (m, n \geq 0)$

- Every $A \in$ Terms that is not immediate is proper

$$\text{PrT} = (\text{Terms} - \text{ImT}) \tag{16}$$

Immediate terms do not carry algorithmic sense:
den$(p(v_1)\dots(v_m))$ is by variable valuation, in memory $p \in$ RecV.

Definition (Congruence Relation, informally)

The *congruence* relation is the smallest equivalence relation (i.e., reflexive, symmetric, transitive) between L^{λ}_{ar}-terms, $A \equiv_{\mathsf{c}} B$, that is closed under:

1. operators of term-formation:
   - application
   - λ-abstraction
   - logic operators
   - pure, logic quantifiers
   - acyclic recursion
   - restriction
2. renaming bound variables (pure and recursion), without causing variable collisions
3. re-ordering of the assignments within the acyclic sequences of assignments in the recursion terms
4. re-ordering of the restriction sub-terms in the restriction terms

[Congruence] If $A \equiv_c B$, then $A \Rightarrow B$ (cong)

[Transitivity] If $A \Rightarrow B$ and $B \Rightarrow C$, then $A \Rightarrow C$ (trans)

[Compositionality]

- If $A \Rightarrow A'$ and $B \Rightarrow B'$, then $A(B) \Rightarrow A'(B')$ (ap-comp)
- If $A \Rightarrow B$, and $\xi \in \{\lambda, \exists, \forall\}$, then $\xi(u)(A) \Rightarrow \xi(u)(B)$ (lq-comp)
- If $A_i \Rightarrow B_i$ $(i = 0, \ldots, n)$, then

$$A_0 \text{ where } \{ p_1 := A_1, \ldots, p_n := A_n \} \Rightarrow B_0 \text{ where } \{ p_1 := B_1, \ldots, p_n := B_n \} \quad \text{(wh-comp)}$$

- If $A_0 \Rightarrow B_0$ and $C_i \Rightarrow R_i$ $(i = 0, \ldots, n)$, then

$$A_0 \text{ such that } \{ C_1, \ldots, C_n \} \Rightarrow B_0 \text{ such that } \{ R_1, \ldots, R_n \} \quad \text{(st-comp)}$$

Reduction Rules (to be continued)

[Head Rule] Given that $p_i \neq q_j$ and no p_i occurs freely in any B_j,

$$\big(A_0 \text{ where } \{\, \overrightarrow{p} := \overrightarrow{A} \,\}\big) \text{ where } \{\, \overrightarrow{q} := \overrightarrow{B} \,\}$$
$$\Rightarrow A_0 \text{ where } \{\, \overrightarrow{p} := \overrightarrow{A},\ \overrightarrow{q} := \overrightarrow{B} \,\} \tag{head}$$

[Bekič-Scott Rule] Given that $p_i \neq q_j$ and no q_i occurs freely in any A_j

$$A_0 \text{ where } \{\, p := \big(B_0 \text{ where } \{\, \overrightarrow{q} := \overrightarrow{B} \,\}\big),\ \overrightarrow{p} := \overrightarrow{A} \,\}$$
$$\Rightarrow A_0 \text{ where } \{\, p := B_0, \overrightarrow{q} := \overrightarrow{B},\ \overrightarrow{p} := \overrightarrow{A} \,\} \tag{B-S}$$

[Recursion-Application Rule] Given that no p_i occurs freely in B,

$$\big(A_0 \text{ where } \{\, \overrightarrow{p} := \overrightarrow{A} \,\}\big)(B)$$
$$\Rightarrow A_0(B) \text{ where } \{\, \overrightarrow{p} := \overrightarrow{A} \,\} \tag{recap}$$

Reduction Rules (to be continued)

[Application Rule] Given that $B \in$ PrT is a proper term, and p is fresh, $p \in \left[\,\text{RecV} - \left(\text{FV}\left(A(B)\right) \cup \text{BV}\left(A(B)\right)\right)\right]$,

$$A(B) \Rightarrow \left[A(p) \text{ where } \{\, p := B \,\}\right] \tag{ap}$$

[λ and Quantifiers rules] Let $\xi \in \{\, \lambda, \exists, \forall \,\}$.
Given fresh $p'_i \in \left[\,\text{RecV} - \left(\text{FV}(A) \cup \text{BV}(A)\right)\right]$, $i = 1, \dots, n$, for $A \equiv A_0$ where $\{\, p_1 := A_1, \dots, p_n := A_n \,\}$ and replacements A'_i in (20):

$$A'_i \equiv \left[A_i\{\, p_1 :\equiv p'_1(u), \dots, p_n :\equiv p'_n(u) \,\}\right] \tag{20}$$

$$\begin{aligned} &\xi(u)\left(A_0 \text{ where } \{\, p_1 := A_1, \dots, p_n := A_n \,\}\right) \\ &\Rightarrow \xi(u)\, A'_0 \text{ where } \{\, p'_1 := \lambda(u)\, A'_1, \dots, p'_n := \lambda(u)\, A'_n \,\} \end{aligned} \tag{ξ}$$

Restriction Rules of $\mathrm{L}_{rar}^{\lambda}$

- each $R_i^{\tau_i} \in$ Terms in $\overrightarrow{R}$ is immediate and $\tau_i \in \{ \mathsf{t}, \widetilde{\mathsf{t}} \}$
- each $C_j^{\tau_j} \in$ Terms is proper and $\tau_j \in \{ \mathsf{t}, \widetilde{\mathsf{t}} \}$ $(j = 1, \dots, m,\ m \geq 0)$
- $a_0, c_j \in$ RecV $(j = 1, \dots, m)$ fresh

(st1) Rule A_0 is an immediate term, $m \geq 1$

$$\begin{aligned} &(A_0 \text{ such that } \{ C_1, \dots, C_m, \overrightarrow{R} \}) \\ \Rightarrow\ &(A_0 \text{ such that } \{ c_1, \dots, c_m, \overrightarrow{R} \}) \\ &\qquad \text{where } \{ c_1 := C_1,\ \dots, c_m := C_m \} \end{aligned} \tag{st1}$$

(st2) Rule A_0 is a proper term

$$\begin{aligned} &(A_0 \text{ such that } \{ C_1, \dots, C_m, \overrightarrow{R} \}) \\ \Rightarrow\ &(a_0 \text{ such that } \{ c_1, \dots, c_m, \overrightarrow{R} \}) \\ &\qquad \text{where } \{ a_0 := A_0, \\ &\qquad\qquad c_1 := C_1,\ \dots, c_m := C_m \} \end{aligned} \tag{st2}$$

γ^*-Reduction — stronger reduction

Definition ($\gamma*$-condition)

A term $A \in$ Terms satisfies the γ^*-condition for an assignment $p := \lambda(\overrightarrow{u}^{\overrightarrow{\sigma}})\lambda(v^{\sigma})P^{\tau} : (\overrightarrow{\sigma} \to (\sigma \to \tau))$, with respect to $\lambda(v^{\sigma})$, iff A is of the form: (23a)–(23c):

$$A \equiv A_0 \text{ where } \{ \overrightarrow{a} := \overrightarrow{A}, \tag{23a}$$

$$p := \lambda(\overrightarrow{u})\lambda(v)P, \tag{23b}$$

$$\overrightarrow{b} := \overrightarrow{B} \} \tag{23c}$$

such that the following holds:

1. $v \notin \mathsf{FreeVars}(P)$
2. All occurrences of p in A_0, $\overrightarrow{A}$, and $\overrightarrow{B}$ are occurrences:
   - in $p(\overrightarrow{u})(v)$,
   - which are in the scope of $\lambda(v)$ modulo renaming the variables $\overrightarrow{u}, v$

(γ^*)-rule

$$
\begin{aligned}
A \equiv A_0 \text{ where } \{ \overrightarrow{a} &:= \overrightarrow{A}, && (24a)\\
p &:= \lambda(\overrightarrow{u})\lambda(v)P, && (24b)\\
\overrightarrow{b} &:= \overrightarrow{B} \} && (24c)\\
\Rightarrow_{(\gamma^*)} A_0' \text{ where } \{ \overrightarrow{a} &:= \overrightarrow{A}', && (24d)\\
p' &:= \lambda(\overrightarrow{u})P, && (24e)\\
\overrightarrow{b} &:= \overrightarrow{B'} \} && (24f)
\end{aligned}
$$

given that:

- $A \in$ Terms satisfies the γ^*-condition (in Definition 3) for $p := \lambda(\overrightarrow{u})\lambda(v)P : (\overrightarrow{\sigma} \to (\sigma \to \tau))$, with respect to $\lambda(v)$
- $p' \in \mathsf{RecV}_{(\overrightarrow{\sigma} \to \tau)}$ is a fresh recursion variable
- $\overrightarrow{X'} \equiv \overrightarrow{X}\{p(\overrightarrow{u})(v) :\equiv p'(\overrightarrow{u})\}$ is the result of the replacements

  $X_i\{p(\overrightarrow{u})(v) :\equiv p'(\overrightarrow{u})\}$,
  i.e., replacing all occurrences of $p(\overrightarrow{u})(v)$ by $p'(\overrightarrow{u})$, in all corresponding parts $X_i \equiv A_i$, $X_i \equiv B_i$, in (24a)–(24f), modulo renaming the variables $\overrightarrow{u}, v$

Theorem (γ^*-Canonical Form Theorem)

For each $A \in \mathsf{Terms}$, *there is a unique up to congruence,* γ^**-irreducible* $\mathsf{cf}_{\gamma^*}(A) \in \mathsf{Terms}$, *s.th.:*

1. *for some explicit,* γ^**-irreducible* $A_0, \ldots, A_n \in \mathsf{Terms}$ $(n \geq 0)$

$$\mathsf{cf}_{\gamma^*}(A) \equiv A_0 \text{ where } \{p_1 := A_1, \ldots, p_n := A_n\}$$

2. $A \Rightarrow^*_{\gamma^*} \mathsf{cf}_{\gamma^*}(A)$
3. *for every* B, *such that* $A \Rightarrow^*_{\gamma^*} B$ *and* B *is* γ^**-irreducible, it holds that* $B \equiv_c \mathsf{cf}_{\gamma^*}(A)$
   i.e., $\mathsf{cf}_{\gamma^*}(A)$ *is unique, up to congruence*
4. $\mathsf{Consts}(\mathsf{cf}_{\gamma^*}(A)) = \mathsf{Consts}(A)$ *and*
5. $\mathsf{FreeV}(\mathsf{cf}_{\gamma^*}(A)) = \mathsf{FreeV}(A)$

Proof.

The proof is by induction on term structure of A, (7a)–(7e), (7f), using reduction rules, definitions, and properties of reduction.
The reduction rules and their applications do not remove and do not add any constants and free variables. □

Algorithmic Semantics of L_{ar}^{λ} / L_{r}^{λ}

How is the algorithmic meaning / semantics of a proper (non-immediate) $A \in$ Terms determined?

- For every term $A \in$ Terms, by the Canonical Form Theorem 4:

$$A \Rightarrow \mathsf{cf}(A)$$

$$A \Rightarrow_{\gamma^*} \mathsf{cf}_{\gamma^*}(A)$$

- For each proper (i.e., non-immediate) $A \in$ Terms, $\mathsf{cf}(A)$ / $\mathsf{cf}_{\gamma^*}(A)$ determines the algorithm $\mathsf{alg}(A)$ for computing $\mathsf{den}(A)$

Theorem (Effective Reduction Calculi)

For every term $A \in$ Terms, its canonical forms $\mathsf{cf}(A)$ and $\mathsf{cf}_{\gamma^*}(A)$ are effectively computed, by the extended reduction calculus.

Definition (of Algorithmic Equivalence / Synonymy)

Two terms $A, B \in$ Terms are algorithmically equivalent, $A \approx B$, in a given semantic structure $\mathfrak{A}$, i.e., referentially synonymous in $\mathfrak{A}$, iff

- A and B are both immediate, or
- A and B are both proper

and there are explicit, irreducible terms (of appropriate types), A_0, ..., A_n, B_0, ..., B_n, $n \geq 0$, such that:

1. $A \Rightarrow_{\mathsf{cf}} A_0$ where $\{ p_1 := A_1, \ldots, p_n := A_n \} \equiv \mathsf{cf}(A)$
2. $B \Rightarrow_{\mathsf{cf}} B_0$ where $\{ p_1 := B_1, \ldots, p_n := B_n \} \equiv \mathsf{cf}(B)$
3. for all $i \in \{ 0, \ldots, n \}$
   a. for every $x \in \mathsf{PureV} \cup \mathsf{RecV}$,
   $$x \in \mathsf{FreeV}(A_i) \quad \text{iff} \quad x \in \mathsf{FreeV}(B_i) \tag{25}$$
   b. $\mathsf{den}(A_i) = \mathsf{den}(B_i)$

Type Theory $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ / L^{λ}_{r} is more expressive than Gallin TY2

Theorem (Moschovakis [13] 2006, §3.24 (mild adjustment))

1. *For any explicit (λ-calculus) $A \in$ Terms, there is no (assignment) memory location, bound via* where *in its canonical form, which occurs in more than one of its parts A_i $(0 \leq i \leq n)$ of* $\mathrm{cf}(A)$ / $\mathrm{cf}_{\gamma^*}(A)$
2. *Assume that $A \in$ Terms is such that an assignment location $p \in$ RecV, bound via* where *in its canonical form* $\mathrm{cf}(A)$ / $\mathrm{cf}_{\gamma^*}(A)$, *occurs in (at least) two assignment parts, and the denotations of those parts depend essentially on p:*
Then, there is no explicit (λ-calculus) term $B \in$ Terms, such that B is algorithmically equivalent to A, $B \approx A$,
i.e., for all λ-calculus $B \in$ Terms, $B \not\approx A$.

The proof is by Moschovakis [13] (2006). I provide it for the extended $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ / L^{λ}_{r}

Reductions with Pure Quantifier Rules: Algorithmic Patterns and Instantiations

- Assume $cube, large_0 \in \mathsf{Consts}_{(\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})}$, in the typical Aristotelian form:

$$\text{Some cube is large} \xrightarrow{\text{render}} B \equiv \exists x(cube(x) \wedge large_0(x)) \tag{26a}$$

$$B \Rightarrow \exists x((c \wedge l) \text{ where } \{\, c := cube(x),\, l := large_0(x)\,\}) \tag{26b}$$

by 2 x (ap) (ap-comp), (recap), (wh-comp), (head), (lq-comp)

$$\Rightarrow \underbrace{\exists x(c'(x) \wedge l'(x))}_{B_0 \text{ algorithmic pattern}} \text{ where } \{ \tag{26c}$$

$$\underbrace{c' := \lambda(x)(cube(x)),\, l' := \lambda(x)(large_0(x))}_{\text{instantiations of memory slots } c',\, l'}\} \equiv \mathsf{cf}(B) \tag{26d}$$

from (26c), by (ξ) to $\exists$

$$\approx \underbrace{\exists x(c'(x) \wedge l'(x))}_{B_0 \text{ algorithmic pattern}} \text{ where } \{ \underbrace{c' := cube,\, l' := large_0}_{\text{instantiations of memory slots } c',\, l'} \} \equiv B' \tag{26e}$$

$$\text{by Def. 6 from (26c)–(26d), } \mathsf{den}(\lambda(x)(cube(x))) = \mathsf{den}(cube), \quad \mathsf{den}(\lambda(x)(large_0(x))) = \mathsf{den}(large_0) \tag{26f}$$

$$\text{Some cube is large} \xrightarrow{\text{render}} T, \quad \mathit{large} \in \mathsf{Consts}_{((\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}) \to (\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}))} \tag{27a}$$

$$T \equiv \exists x \big[\mathit{cube}(x) \wedge \underbrace{\mathit{large}(\mathit{cube})(x)}_{\text{by predicate modification}} \big] \Rightarrow \ldots \tag{27b}$$

$$\Rightarrow \exists x \big[(c_1 \wedge l) \text{ where } \{ c_1 := \mathit{cube}(x), \tag{27c}$$

$$l := \mathit{large}(c_2)(x),\ c_2 := \mathit{cube} \}\big] \tag{27d}$$

$$\Rightarrow \exists x (c_1'(x) \wedge l'(x)) \text{ where } \{ c_1' := \lambda(x)(\mathit{cube}(x)), \tag{27e}$$

$$l' := \lambda(x)(\mathit{large}(c_2'(x))(x)),\ c_2' := \lambda(x)\mathit{cube} \} \tag{27f}$$

$$\equiv \mathsf{cf}(T) \qquad \text{(27e)–(27f) is by } (\xi) \text{ on (27c)–(27d)}$$

$$\Rightarrow_{\gamma^*} \exists x (c_1'(x) \wedge l'(x)) \text{ where } \{ c_1' := \lambda(x)(\mathit{cube}(x)), \tag{27g}$$

$$l' := \lambda(x)(\mathit{large}(c_2)(x)),\ c_2 := \mathit{cube} \} \tag{27h}$$

$$\equiv \mathsf{cf}_{\gamma^*}(T)$$

$$\approx \exists x (c_1'(x) \wedge l'(x)) \text{ where } \{ c_1' := \mathit{cube}, \tag{27i}$$

$$l' := \lambda(x)(\mathit{large}(c_2)(x)),\ c_2 := \mathit{cube} \} \tag{27j}$$

$$\text{Some cube is large} \xrightarrow{\text{render}} C, \quad large \in \mathsf{Consts}_{((\widetilde{\mathsf{e}}\to\widetilde{\mathsf{t}})\to(\widetilde{\mathsf{e}}\to\widetilde{\mathsf{t}}))}$$

$$C \equiv \underbrace{\exists x\big[c'(x) \land large(c')(x)\big]}_{E_0} \text{ where } \{\, c' := cube \,\} \tag{28a}$$

$$\Rightarrow \underbrace{\exists x\big[\big(c'(x) \land l\big) \text{ where } \{\, l := large(c')(x) \,\}\big]}_{E_1} \text{ where } \{\, c' := cube \,\} \tag{28b}$$

from (28a), by (ap) to $\land$ of E_0; (lq-comp); (wh-comp)

$$\Rightarrow \big[\underbrace{\exists x\big(c'(x) \land l'(x)\big) \text{ where } \{\, l' := \lambda(x)\big(large(c')(x)\big) \,\}}_{E_2}\big] \text{ where } \{\, c' := cube \,\} \tag{28c}$$

from (28b), by (ξ) to $\exists$

$$\Rightarrow \underbrace{\exists x\big(c'(x) \land l'(x)\big)}_{C_0 \text{ an algorithmic pattern}} \text{ where } \{\underbrace{c' := cube,\ l' := \lambda(x)\big(large(c')(x)\big)}_{\text{instantiations of memory } c',\, l'}\} \equiv \mathsf{cf}(C) \tag{28d}$$

from (28c), by (head); (cong)

Proposition

1. *The L^{λ}_{ar}-terms $C \approx \mathrm{cf}(C)$ in (28a)–(28d), and many other L^{λ}_{ar}-terms, are not algorithmically equivalent to any explicit terms*
2. L^{λ}_{ar} *is a strict, proper extension of TY_2, Gallin [4]*
3. *and of a la Montague semantics via inclusion of Montague IL in TY_2*

Outline of a proof:
(1) follows by Theorem 7
(2) follows by Theorem 7, and (1)
(3) Gallin [4] provides an interpretation of Montague IL [14] into TY_2.
Suitable interpretation can be given directly in L^{λ}_{ar} (L^{λ}_{r}).

Placement of L^{λ}_{ar} in a class of type theories

$$\text{Montague IL} \subsetneq \text{Gallin TY}_2 \subsetneq \text{Moschovakis } L^{\lambda}_{ar} \subsetneq \text{Moschovakis } L^{\lambda}_{r} \tag{29}$$

Generalised Two-Argument Quantifiers: $Q : ((\widetilde{e} \to \widetilde{t}) \to ((\widetilde{e} \to \widetilde{t}) \to \widetilde{t}))$

$$\text{some, every} \xrightarrow{\text{render}} \mathit{some}, \mathit{every} \in \mathsf{Consts}_{[(\widetilde{e}\to\widetilde{t})\to((\widetilde{e}\to\widetilde{t})\to\widetilde{t})]} \tag{30}$$

$$[\text{some}_{\mathrm{DET}}\ \text{cube}_{\mathrm{N}}]_{\mathrm{NP}} \xrightarrow{\text{render}} \mathit{some}(\mathit{cube}) : ((\widetilde{e} \to \widetilde{t}) \to \widetilde{t}) \tag{31}$$

$$\Rightarrow_{\mathsf{cf}} \big[\mathit{some}(d) \text{ where } \{\, d := \mathit{cube} \,\}\big] \tag{32}$$

$$\text{Some cube is large} \xrightarrow{\text{render}} A_0/A_1/A_2 \quad \text{(options)} \tag{33a}$$

$$A_0 \equiv \big(\mathit{some}(\mathit{cube})\big)(\mathit{large}_0) : \widetilde{t} \qquad \text{typical } \lambda\text{-term} \tag{33b}$$

$$\Rightarrow_{\mathsf{cf}} \underbrace{\mathit{some}(p_1)(p_2) \ \text{ where } \{p_1 := \mathit{cube},\ p_2 := \mathit{large}_0\}}_{\text{recursion term}} \tag{33c}$$

$$A_1 \equiv \mathit{some}(p_1)(p_2) \text{ where } \{p_1 := \mathit{cube},\ p_2 := \mathit{large}(p_1)\} \tag{33d}$$

$$A_2 \equiv \underbrace{Q(p_1)(p_2)}_{\text{alg. pattern}} \text{ where } \{\underbrace{Q := \mathit{some},\ p_1 := \mathit{cube},\ p_2 := \mathit{large}(p_1)}_{\text{instantiations of memory}}\} \tag{33e}$$

Alternatives: $Q := \mathit{every}$, $Q := \mathit{one}$, $Q := \mathit{two}$, $Q := \mathit{most}$, etc.
No explicit terms are algorithmically equivalent to A_1 and A_2, by Th. 7.

$$[K\ [\text{is}\ [\text{larger}_{\text{ADJ}}\ \text{than}\ [\text{some}_{\text{DET}}\ \text{number}_{\text{N}}]_{\text{NP}}]_{\text{ADJP}}]_{\text{VP}}]_{\text{S}} \xrightarrow{\text{render}} A \tag{34a}$$

$$A \equiv \Big[\lambda y \big[\big[some(number)\big]\big(\lambda x_d\, larger(x_d)(y)\big)\big]\Big](K) \Rightarrow \quad \dots \tag{34b}$$

$$\Rightarrow \Big[\lambda(y_k)\Big(some\big(d'(y_k)\big)\big(h(y_k)\big)\Big) \text{ where } \{\, d' := \lambda(y_k) number,\ h := \lambda(y_k)\lambda(x_d) larger(x_d)(y_k)\,\}\Big](K) \tag{34c}$$

$$\Rightarrow_{\mathsf{cf}}\ \mathsf{cf}(A) \equiv \tag{34d}$$

$$\Big[\lambda(y_k)\Big(some\big(d'(y_k)\big)\big(h(y_k)\big)\Big)\Big](k) \text{ where } \{\, h := \lambda(y_k)\lambda(x_d) larger(x_d)(y_k),\ d' := \lambda(y_k) number,\ k := K\,\} \tag{34e}$$

$$\Rightarrow_{\gamma^*}\ \big[\lambda(y_k) some(d)\big(h(y_k)\big)\big](k) \text{ where } \{\, h := \lambda(y_k)\lambda(x_d) larger(x_d)(y_k),\ d := number,\ k := K\,\} \tag{34f}$$

$$[K \; [\text{is} \; [\text{larger}_{\mathrm{ADJ}} \; \text{than} \; [\text{some}_{\mathrm{DET}} \; \text{number}_{\mathrm{N}}]_{\mathrm{NP}}]_{\mathrm{ADJP}}]_{\mathrm{VP}}]_{\mathrm{S}} \xrightarrow{\text{render}} A_3 \tag{35a}$$

$$A_3 \equiv \Big[\lambda y_k \big[\big[Q(number)\big]\big(\lambda x_d \, larger(x_d)(y_k)\big) \text{ where } \{ Q := some\}\big]\Big](K) \Rightarrow \ \dots \tag{35b}$$

$$\Rightarrow \Big[\lambda(y_k)\Big(Q\big(d'(y_k)\big)\big(h(y_k)\big)\Big) \text{ where } \{\, Q := some, \ d' := \lambda(y_k) number, \ h := \lambda(y_k)\lambda(x_d) larger(x_d)(y_k)\,\}\Big](K) \tag{35c}$$

$$\Rightarrow_{\mathsf{cf}} \ \mathsf{cf}(A) \equiv \tag{35d}$$

$$\Big[\lambda(y_k)\Big(Q\big(d'(y_k)\big)\big(h(y_k)\big)\Big)\Big](k) \text{ where } \{\, Q := some, \ h := \lambda(y_k)\lambda(x_d) larger(x_d)(y_k), \ d' := \lambda(y_k) number, \ k := K\,\} \tag{35e}$$

$$\Rightarrow_{\gamma^*} \ \big[\lambda(y_k) Q(d)\big(h(y_k)\big)\big](k) \text{ where } \{\, Q := some, \ h := \lambda(y_k)\lambda(x_d) larger(x_d)(y_k), \ d := number, \ k := K\,\} \tag{35f}$$

de dicto and *de re* renderings of quantifiers shared algorithmic pattern

$$\text{Every cube is larger than some dodeca} \xrightarrow{\text{render}} \qquad \text{(de dicto)}$$

$$R_3 \text{ where } \{R_3 := every(p)(R_2), \tag{36a}$$

$$R_2 := \lambda(x_2) some(b)(R_1(x_2)), \tag{36b}$$

$$R_1 := \lambda(x_2)\lambda(x_1) larger(x_1)(x_2), \tag{36c}$$

$$p := cube,\ b := dodeca\} \tag{36d}$$

$$\text{Every cube is larger than some dodeca} \xrightarrow{\text{render}} \qquad \text{(de re)}$$

$$R_3 \text{ where } \{R_3 := some(b)(R_1), \tag{37a}$$

$$R_1 := \lambda(x_1) every(p)(R_2(x_1)), \tag{37b}$$

$$R_2 := \lambda(x_1)\lambda(x_2) larger(x_1)(x_2), \tag{37c}$$

$$p := cube,\ b := dodeca\} \tag{37d}$$

de dicto term S_{21}

$$
\begin{aligned}
S_{21} \equiv R_3 \text{ where } \{\, R_3 &:= Q_2(R_2), && \text{(38a)}\\
R_2 &:= \lambda(x_2) Q_1(R_1^1(x_2)), && \text{(38b)}\\
R_1^1 &:= \lambda(x_2)\lambda(x_1) h(x_1)(x_2), && \text{(38c)}\\
Q_1 &:= q_1(d_1),\ Q_2 := q_2(d_2), && \text{(38d)}\\
q_2 &:= \mathit{every},\ d_2 := \mathit{cube}, && \text{(38e)}\\
q_1 &:= \mathit{some},\ d_1 := \mathit{dodeca},\ h := \mathit{larger}\,\} && \text{(38f)}
\end{aligned}
$$

de re term S_{12}

$$
\begin{aligned}
S_{12} \equiv R_3 \text{ where } \{\, R_3 &:= Q_1(R_1), && \text{(39a)}\\
R_1 &:= \lambda(x_1) Q_2(R_2^1(x_1)), && \text{(39b)}\\
R_2^1 &:= \lambda(x_1)\lambda(x_2) h(x_1)(x_2), && \text{(39c)}\\
Q_1 &:= q_1(d_1),\ Q_2 := q_2(d_2), && \text{(39d)}\\
q_2 &:= \mathit{every},\ d_2 := \mathit{cube}, && \text{(39e)}\\
q_1 &:= \mathit{some},\ d_1 := \mathit{dodeca},\ h := \mathit{larger}\,\} && \text{(39f)}
\end{aligned}
$$

Constrained Underspecified Terms

$$
\begin{aligned}
U \equiv R_3 \text{ where } \{\, & l_1 := Q_1(R_1),\ l_2 := Q_2(R_2), && \text{(40a)}\\
& Q_1 := q_1(d_1),\ Q_2 := q_2(d_2), && \text{(40b)}\\
& q_1 := \mathit{some},\ q_2 := \mathit{every}, && \text{(40c)}\\
& h := \mathit{larger},\ d_1 := \mathit{dodeca},\ d_2 := \mathit{cube} \,\} && \text{(40d)}\\
\text{s.t. } \{\, & Q_i \text{ binds the } i\text{-th argument of } h, && \text{(40e)}\\
& R_3 \text{ binds (dominates) each } Q_i\ (i = 1, 2) \,\} && \text{(40f)}
\end{aligned}
$$

- U is underspecified (per se), but restricted: R_3, $R_i (i = 1, 2)$ are free, restricted recursion variables
- any of its specifications have to satisfy the constraints

U can be specified to *de dicto* term:

$$\begin{aligned}
U_{21} \equiv R_3 \text{ where } \{\, & R_3 := l_2,\ l_2 := Q_2(R_2), && \text{(41a)}\\
& R_2 := \lambda(x_2) l_1^1(x_2),\ l_1^1 := \lambda(x_2) Q_1(R_1^1(x_2)), && \text{(41b)}\\
& R_1^1 := \lambda(x_2)\lambda(x_1) h(x_1)(x_2), && \text{(41c)}\\
& Q_1 := q_1(d_1),\ Q_2 := q_2(d_2), && \text{(41d)}\\
& q_2 := \mathit{every},\ d_2 := \mathit{cube}, && \text{(41e)}\\
& q_1 := \mathit{some},\ d_1 := \mathit{dodeca},\ h := \mathit{larger}\,\} && \text{(41f)}
\end{aligned}$$

U_{21} can be simplified to the similar, not algorithmically synonymous term:

$$\begin{aligned}
S_{21} \equiv R_3 \text{ where } \{\, & R_3 := Q_2(R_2), && \text{(42a)}\\
& R_2 := \lambda(x_2) Q_1(R_1^1(x_2)), && \text{(42b)}\\
& R_1^1 := \lambda(x_2)\lambda(x_1) h(x_1)(x_2), && \text{(42c)}\\
& Q_1 := q_1(d_1),\ Q_2 := q_2(d_2), && \text{(42d)}\\
& q_2 := \mathit{every},\ d_2 := \mathit{cube}, && \text{(42e)}\\
& q_1 := \mathit{some},\ d_1 := \mathit{dodeca},\ h := \mathit{larger}\,\} && \text{(42f)}
\end{aligned}$$

U can be specified to the *de re* term:

$$U_{12} \equiv R_3 \text{ where } \{\, R_3 := l_1,\ l_1 := Q_1(R_1), \tag{43a}$$
$$R_1 := \lambda(x_1) l_2^1(x_1),\ l_2^1 := \lambda(x_1) Q_2(R_2^1(x_1)), \tag{43b}$$
$$R_2^1 := \lambda(x_1)\lambda(x_2) h(x_1)(x_2), \tag{43c}$$
$$Q_1 := q_1(d_1),\ Q_2 := q_2(d_2), \tag{43d}$$
$$q_2 := every,\ d_2 := cube, \tag{43e}$$
$$q_1 := some,\ d_1 := dodeca,\ h := larger \,\} \tag{43f}$$

U_{12} can be simplified to the similar, not algorithmically synonymous term:

$$S_{12} \equiv R_3 \text{ where } \{\, R_3 := Q_1(R_1), \tag{44a}$$
$$R_1 := \lambda(x_1) Q_2(R_2^1(x_1)), \tag{44b}$$
$$R_2^1 := \lambda(x_1)\lambda(x_2) h(x_1)(x_2), \tag{44c}$$
$$Q_1 := q_1(d_1),\ Q_2 := q_2(d_2), \tag{44d}$$
$$q_2 := every,\ d_2 := cube, \tag{44e}$$
$$q_1 := some,\ d_1 := dodeca,\ h := larger \,\} \tag{44f}$$

$$\Phi \equiv \text{The cube is large} \tag{45}$$

- First Order Logic (FOL) A

$$\Phi \xrightarrow{\text{render}} A \equiv \exists x \big[\underbrace{\forall y(cube(y) \leftrightarrow x = y)}_{uniqueness} \ \wedge \ isLarge(x)\big] \tag{46a}$$

$$S \equiv \exists x \big[\underbrace{\forall y(P(y) \leftrightarrow x = y)}_{uniqueness} \ \wedge \ Q(x)\big] \tag{46b}$$

In FOL, A in (46a) has the following features:

- Existential quantification as the direct, topmost predication
- Uniqueness of the existing entity
- There is no referential force to the object denoted by the descriptor NP: $[\text{the cube}]_{\text{NP}}$
- There is no compositional analysis, i.e., no "derivation", of A from the components of Φ

- Higher Order Logic (HOL): Henkin (1950) and Mostowski (1957)
  a significant, positive step; but lost referential force

$$\text{the} \xrightarrow{\text{render}} T \equiv \left[\lambda P \lambda Q \left[\exists x \left[\underbrace{\forall y (P(y) \leftrightarrow x = y)}_{uniqueness} \;\wedge\; Q(x)\right]\right]\right] \quad \text{(47a)}$$

$$\text{the cube} \xrightarrow{\text{render}} C \equiv T(cube)$$

$$C \equiv \left[\lambda P \lambda Q \left[\exists x \left[\underbrace{\forall y (P(y) \leftrightarrow x = y)}_{uniqueness} \;\wedge\; Q(x)\right]\right]\right](cube) \quad \text{(47b)}$$

$$\models\!\!\!\!\dashv \; D \equiv \lambda Q \left[\exists x \left[\underbrace{\forall y (cube(y) \leftrightarrow x = y)}_{uniqueness} \;\wedge\; Q(x)\right]\right] \quad \text{(47c)}$$

(fr. (47b) by β-reduction)

$$\Phi \equiv \text{The cube is large} \xrightarrow{\text{render}} B \equiv D(isLarge) \quad \text{(48a)}$$

$$B \equiv \left[\lambda Q \left[\exists x \left[\underbrace{\forall y (cube(y) \leftrightarrow x = y)}_{uniqueness} \;\wedge\; Q(x)\right]\right]\right](isLarge) \quad \text{(48b)}$$

$$\models\!\!\!\!\dashv \; \exists x \left[\underbrace{\forall y (cube(y) \leftrightarrow x = y)}_{uniqueness} \;\wedge\; isLarge(x)\right] \quad \text{(48c)}$$

(fr. (48b) by β-reduction)

Example: rendering of the definite article "the" — Option 1

- Rendering the definite article "the" to a constant:

$$\text{the} \xrightarrow{\text{render}} \mathit{the} \in \mathsf{Consts}_{((\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}}) \to \widetilde{\mathsf{e}})} \tag{49}$$

- together with the following denotation of the constant the, requiring "uniqueness" of the denoted object:

$$\big[\big(\mathsf{den}(\mathit{the})\big)(g)\big](\bar{p})(s_0) = \begin{cases} y, & \text{if } y \text{ is the unique } y \in \mathbb{T}_{\mathsf{e}}, \\ & \ \text{for which } \bar{p}(s \mapsto y)(s_0) = 1 \\ \mathsf{er}, & \text{otherwise} \\ & \ \text{i.e., there is no unique entity} \\ & \ \text{that has the property } \bar{p} \text{ in } s_0 \end{cases} \tag{50}$$

for every $\bar{p} \in \mathbb{T}_{(\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})}$ and every $s_0 \in \mathbb{T}_{\mathsf{s}}$

There are other possibilities for rendering the definite article "the", e.g., see Loukanova [10].

Option 3: the definite determiner "the" and descriptors: Underspecification

We can render "the" to A_1 or $\mathsf{cf}(A_1)$, underspecified for p:

$$\text{the} \xrightarrow{\text{render}} A_1 \equiv \big(q \text{ s.t. } \{\, unique(p)(q) \,\}\big) : \widetilde{\mathsf{e}} \tag{51a}$$

$$\text{the} \xrightarrow{\text{render}} \mathsf{cf}(A_1) \equiv \big(q \text{ s.t. } \{\, U \,\}\big) \text{ where } \{\, U := unique(p)(q) \,\} \tag{51b}$$

$$p \in \mathsf{RecV}_{(\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})}, \quad q \in \mathsf{RecV}_{\widetilde{\mathsf{e}}} \tag{51c}$$

- q is the object, whoever it turns to be, by having the property $unique(p)$, by $unique(p)(q)$

$$\text{the cube} \xrightarrow{\text{render}} \mathsf{cf}(A_2) : \widetilde{\mathsf{e}} \tag{52a}$$

$$A_2 \equiv \big(q \text{ s.t. } \{\, unique(p)(q) \,\}\big) \text{ where } \{\, p := cube \,\} \tag{52b}$$

$$\Rightarrow_{\mathsf{cf}} \mathsf{cf}(A_2) \equiv \big(q \text{ s.t. } \{\, U \,\}\big) \text{ where } \{\, U := unique(p)(q), \; p := cube \,\} \tag{52c}$$

by (st1), (head), from (52b)

$$\text{The cube is large} \xrightarrow{\text{render}} \mathsf{cf}(A_3) : \widetilde{\mathsf{t}} \tag{53a}$$

$$A_3 \equiv large(p)\Big(\big(q \text{ s.t. } \{\, unique(p)(q) \,\}\big) \text{ where } \{\, p := cube \,\}\Big) \tag{53b}$$

$$\begin{aligned} &\Rightarrow large(p)(Q) \text{ where } \{ \\ &\qquad Q := \big[\big(q \text{ s.t. } \{\, unique(p)(q) \,\}\big) \text{ where } \{\, p := cube \,\}\big] \,\} \end{aligned} \tag{53c}$$

by (ap), from (53b)

$$\begin{aligned} \Rightarrow_{\mathsf{cf}} \mathsf{cf}(A_3) \equiv large(p)(Q) \text{ where } \{\, & Q := (q \text{ s.t. } \{\, U \,\}), \\ & U := unique(p)(q),\ p := cube \,\} \end{aligned} \tag{53d}$$

by (st1), (wh-comp), (B-S), from (52c), (53c)

Algorithmic Pattern: definite descriptors in predicative statements: Opt3

$$A \equiv L(Q) \text{ where } \{\, Q := (q \text{ s.t. } \{\, U \,\}),\ U := unique(p)(q) \,\} \tag{54a}$$

$$p, q, L \in \mathsf{FreeV}(A),\ p \in \mathsf{RecV}_{(\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})},\ q \in \mathsf{RecV}_{\widetilde{\mathsf{e}}}, \tag{54b}$$

$$Q \in \mathsf{RecV}_{\widetilde{\mathsf{e}}},\ U \in \mathsf{RecV}_{\widetilde{\mathsf{t}}},\ L \in \mathsf{RecV}_{(\widetilde{\mathsf{e}} \to \widetilde{\mathsf{t}})} \tag{54c}$$

$$\text{The number } n \text{ is odd} \xrightarrow{\text{render}} \text{cf}(A_4) : \widetilde{\text{t}} \tag{55a}$$

$$\begin{aligned} A_4 \equiv isOdd\Big(\big(q \text{ s.t. } \{\, unique(N)(q),\ p(q)\,\}\big) \text{ where } \{ \\ q := n,\ p := number,\ N := named\text{-}n\,\}\Big) \end{aligned} \tag{55b}$$

$$\begin{aligned} \Rightarrow_{\text{cf}}\ \text{cf}(A_4) \equiv isOdd(Q) \text{ where } \{\, & Q := \big(q \text{ s.t. } \{\, U, C\,\}\big), \\ & U := unique(N)(q),\ C := p(q), \\ & q := n,\ p := number,\ N := named\text{-}n\,\} \end{aligned} \tag{55c}$$

- direct reference, by assignment; uniqueness and existence are consequences

$$\text{The number } n \text{ is large} \xrightarrow{\text{render}} \text{cf}(A_5) : \widetilde{\text{t}} \tag{56a}$$

$$\begin{aligned} A_5 \equiv isOdd\Big(\big(q \text{ s.t. } \{\, p(q)\,\}\big) \text{ where } \{ \\ q := n,\ p := number\,\}\Big) \end{aligned} \tag{56b}$$

$$\begin{aligned} \Rightarrow_{\text{cf}}\ isOdd(Q) \text{ where } \{\, & Q := \big(q \text{ s.t. } \{\, C\,\}\big),\ C := p(q), \\ & q := n,\ p := number\,\} \end{aligned} \tag{56c}$$

Predication via Coordination: e.g., a class of coordinated Vs, VPs, etc.

$$[\Phi_j]_{\mathrm{NP}} \ \big[[\Theta_L \text{ and } \Psi_H] \ [W_w]_{\mathrm{NP}}\big]_{\mathrm{VP}} \tag{57a}$$

$$\xrightarrow{\text{render}} \underbrace{\lambda x_j \big[\lambda y_w \big(L(x_j)(y_w) \land H(x_j)(y_w)\big)(w)\big](j)}_{\text{algorithmic pattern with memory parameters } L,\ H,\ w,\ j} \tag{57b}$$

$$[\text{The cube}]_j \ [\text{ is larger than and is next to } [\ [\text{its}]_j \text{ predecessor}]_w\ ] \xrightarrow{\text{render}} A \tag{58}$$

$$A \equiv \lambda x_j \big[\lambda y_w \big(\mathit{larger}(y_w)(x_j) \land \mathit{nextTo}(y_w)(x_j)\big) \\ (\mathit{predecessor}(x_j))\big](\mathit{the}(\mathit{cube})) \tag{59a}$$

$$\Rightarrow_{\gamma^*} \underbrace{\lambda x_j \big[\lambda y_w \big(L''(x_j)(y_w) \land H''(x_j)(y_w)\big)(w'(x_j))\big](j)}_{\text{algorithmic pattern with memory parameters } L'',\ H'',\ w',\ j} \tag{59b}$$

$$\text{where } \{\, L'' := \lambda x_j \lambda y_w\, \mathit{larger}(y_w)(x_j), \\ H'' := \lambda x_j \lambda y_w\, \mathit{nextTo}(y_w)(x_j), \\ \underbrace{w' := \lambda x_j \mathit{predecessor}(x_j),\ j := \mathit{the}(c),\ \ c := \mathit{cube}}_{\text{instantiations of memory } L'',\ H'',\ w',\ j} \,\} \tag{59c}$$

- The sentence (60a)–(60b) is a conjunction of propositions, i.e., propositional conjunction

- The computational semantics of (60a)–(60b) can be represented by $\mathsf{cf}_{\gamma^*}(B)$, in (61a)–(61b):

$$\text{[The cube]}_j \text{ is larger than [[its]}_j \text{ predecessor]}_w \tag{60a}$$

$$\text{and [it]}_j \text{ is next to [it]}_w \xrightarrow{\text{render}} B \tag{60b}$$

$$B \equiv \big[\mathit{larger}(w)(j) \land \mathit{nextTo}(w)(j) \big] \text{ where } \{ \\ j := \mathit{the}(\mathit{cube}),\ w := \mathit{predecessor}(j) \} \tag{61a}$$

$$\Rightarrow_{\mathsf{cf}_{\gamma^*}} \big[L \land H \big] \text{ where } \{ L := \mathit{larger}(w)(j),\ H := \mathit{nextTo}(w)(j), \\ w := \mathit{predecessor}(j), \\ j := \mathit{the}(c),\ c := \mathit{cube} \} \tag{61b}$$

Computational Syntax-Semantics of NL by using L_{ar}^{λ} in GCBLG

For syntax-semantics interfaces of Natural Language (NL), I employ:

- Generalised Constraint-Based Lexicalized Grammar (GCBLG), see [7] GCBLG covers a variety of computational grammars, by representing major, common syntactic characteristics of a class of approaches to computational grammar, e.g.:
    - Head-Driven Phrase Structure Grammar (HPSG) [3]
    - Lexical Functional Grammar (LFG) [1]
    - Categorial Grammar (CG) [2, 11]
    - Grammatical Framework (GF) [5] (tentatively)

Computational Syntax-Semantics of NL by using $\mathrm{L}^{\lambda}_{\mathrm{ar}}$ in GCBLG

Generalised Constraint-Based Lexicalized Grammar (GCBLG) covers major syntactic categories of natural language, by linguistically motivated generalizations.

- The syntactic information is distributed among a hierarchy of types
- typed feature-value descriptions: Feature-Value Logics; Attribute-Value (ATV) Matrices
- The semantic representation in syntax-semantics composition and interface, is by the feature SEM and its recursive values

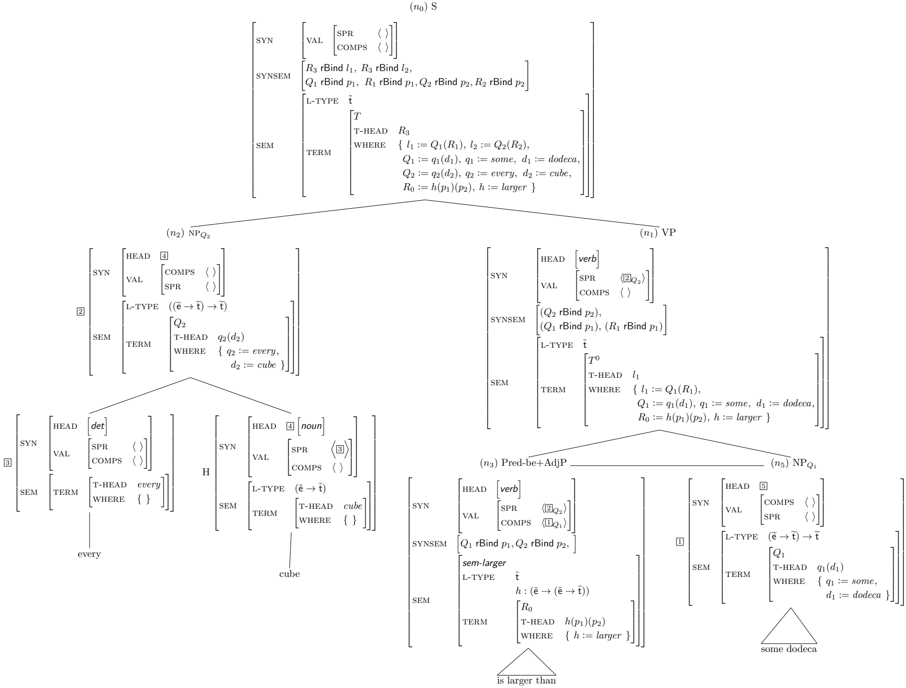
  SEM has typed values that encode recursion terms of $\mathrm{L}^{\lambda}_{\mathrm{ar}}$, alternatively, of DTTSitInfo
- Efficient and effective, computational rendering of NL expressions to γ^*-canonical forms, see Loukanova [6, 9, 8]

Computational Syntax-Semantics of NL by using $\mathrm{L}_{\mathrm{ar}}^{\lambda}$ in GCBLG

Computational Grammar with Syntax-Semantics and Underspecification

For a given NL expression ϕ, its grammar analysis Φ, includes syntax-semantics interface, throughout its constituents

$$\Phi \xrightarrow{\text{render}} A \equiv \mathsf{cf}_{\gamma}(A) \tag{62}$$



(n_0) S
(n_2) NP_{Q_2}
(n_1) VP
every
cube
(n_3) Pred-be+AdjP
(n_5) NP_{Q_1}
is larger than
some dodeca

Motivation for Type Theory L^{λ}_{ar} and Outlook

- L^{λ}_{ar} provides Computational Semantics with:
  - greater semantic distinctions than type-theoretic semantics by λ-calculi, e.g., Montagovian grammars
- L^{λ}_{ar} provides Parametric Algorithms
  Parameters can be instantiated depending on:
  - classes and sets of specific names, NPs, verbs, properties, relations, etc.
  - representing major semantic ambiguities and underspecification [6], at the object level of its formal language, without meta-language variables
- L^{λ}_{ar} with logical operators and pure quantifiers can be used for:
  - proof-theoretic computational semantics and reasoning
  - inferences of semantic information
  - Canonical forms can be used by automatic provers and proof assistants

Looking Forward!

Outlook1: Development of Computational Theories and Applications

- Generalised Computational Grammar: CompSynSem interfaces in NL, HL (human language)
  - Hierarchical lexicon with morphological structure and lexical rules
  - Syntax of NL expressions (phrasal and grammatical dependences)
  - Syntax-semantics inter-relations in lexicon and phrases
- A Big Picture — simplified and approximated, but realistic:

Algorithmic CompSynSem of Human Language (HL)

$$\underbrace{\text{HL Syn} \Longleftrightarrow \underbrace{L_{ar}^{\lambda} \,/\, L_{r}^{\lambda} \,/ SitI \xrightarrow{\text{Reduction Calc}} \text{Canonical Forms}}_{\text{Canonical Computations}} \Longrightarrow \text{Denotations}}_{\text{(Canonically) Algorithmic CompSynSem Interfaces}}$$

(I've done quite a lot of it, but still a lot to do!)

Outlook2: Applications to Human / Natural Language Processing (NLP)

Translations via Algorithmic Syntax-Semantics Interfaces (CompSynSem) Human Languages, Ontologies, and L_{ar}^{λ} / SitI

$$\begin{array}{ccccc}
\text{Lexicon of } \mathsf{L}_0 \iff \text{Syn of } \mathsf{L}_0 & \xleftrightarrow[\text{render}^{-1}]{\text{render}} & L_{ar}^{\lambda}\ \text{/SitI Canonical Terms} \\
 & & \downarrow\uparrow \\
 & & \text{possible} \\
\text{Data / Ontologies / Tree Banks, etc.} & \xleftrightarrow[\text{render}^{-1}]{\text{render}} & \text{modifications} \\
 & & \text{of the terms} \\
 & & \downarrow\uparrow \\
\{\text{Lexicon of } \mathsf{L}_i \iff \text{Syn of } \mathsf{L}_i & \xleftrightarrow[\text{render}^{-1}]{\text{render}} & L_{ar}^{\lambda}\ \text{/SitI Canonical Terms} \\
\mid 1 \leq i \leq n\} & &
\end{array}$$

Some References I

Bresnan, J.: Lexical-Functional Syntax.
Blackwell Publishers, Oxford (2001)

Buszkowski, W.: Mathematical Linguistics and Proof Theory.
In: J. van Benthem, A. ter Meulen (eds.) Handbook of Logic and Language, pp. 683–736. North-Holland, Amsterdam (1997).
DOI https://doi.org/10.1016/B978-044481714-3/50016-3.
URL https://www.sciencedirect.com/science/article/pii/B9780444817143500163

DELPH-IN: Deep Linguistic Processing with HPSG (DELPH-IN) (2018, edited).
URL http://moin.delph-in.net.
Accessed 20-Aug-2023

Some References II

- Gallin, D.: Intensional and Higher-Order Modal Logic: With Applications to Montague Semantics.
  North-Holland Publishing Company, Amsterdam and Oxford, and American Elsevier Publishing Company (1975).
  URL https://doi.org/10.2307/2271880

- The Grammatical Framework GF.
  http://www.grammaticalframework.org.
  Accessed 20-Aug-2023

- Loukanova, R.: Relationships between Specified and Underspecified Quantification by the Theory of Acyclic Recursion.
  ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal **5**(4), 19–42 (2016).
  DOI 10.14201/ADCAIJ201654.
  URL https://doi.org/10.14201/ADCAIJ2016541942

Some References III

- Loukanova, R.: An Approach to Functional Formal Models of Constraint-Based Lexicalized Grammar (CBLG).
  Fundamenta Informaticae **152**(4), 341–372 (2017).
  DOI 10.3233/FI-2017-1524.
  URL https://doi.org/10.3233/FI-2017-1524

- Loukanova, R.: Gamma-Reduction in Type Theory of Acyclic Recursion.
  Fundamenta Informaticae **170**(4), 367–411 (2019).
  URL https://doi.org/10.3233/FI-2019-1867

Some References IV

- Loukanova, R.: Gamma-Star Canonical Forms in the Type-Theory of Acyclic algorithms.
  In: J. van den Herik, A.P. Rocha (eds.) Agents and Artificial Intelligence, *Lecture Notes in Computer Science*, vol. 11352, pp. 383–407. Springer International Publishing, Cham (2019).
  URL https://doi.org/10.1007/978-3-030-05453-3_18

- Loukanova, R.: Restricted Computations and Parameters in Type-Theory of Acyclic Recursion.
  ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal **12**(1), 1–40 (2023).
  URL https://doi.org/10.14201/adcaij.29081

Some References V

- Moortgat, M.: Categorial Type Logics.
  In: J. van Benthem, A. ter Meulen (eds.) Handbook of Logic and Language, pp. 93–177. Elsevier, Amsterdam (1997).
  URL `https://doi.org/10.1016/B978-044481714-3/50005-9`

- Moschovakis, Y.N.: The formal language of recursion.
  Journal of Symbolic Logic **54**(4), 1216–1252 (1989).
  URL `https://doi.org/10.1017/S0022481200041086`

- Moschovakis, Y.N.: A Logical Calculus of Meaning and Synonymy.
  Linguistics and Philosophy **29**(1), 27–89 (2006).
  URL `https://doi.org/10.1007/s10988-005-6920-7`

- Thomason, R.H. (ed.): Formal Philosophy: Selected Papers of Richard Montague.
  Yale University Press, New Haven, Connecticut (1974)