Variable Polyadicity Without Events: A Type-Theoretic Analysis of Event Semantics*

Zhaohui Luo [罗朝晖]
Royal Holloway, U of London

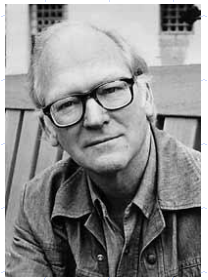
Yunbao Shi [石运宝]
West Anhui University

* Work based on a paper of the above title in 2025.

Theme (of this talk)

- Variable Polyadicity (VP) [论元数量可变性]
  - Key problem solved by Davidsonian event semantics
  - Seemingly unresolvable in set theory/traditional logic
- VP problem is solvable in type theory!
  - Why? [Part I]
  - How? [Part II]
- Our paper: analysis of event semantics in general (omitted in this talk)

Event semantics and the VP problem



- Event semantics [Davidson 1967, Parsons 1990]
  - Popular approach to formal semantics
  - set-theoretic (simple type theory as intermediate)
- Variable Polyadicity (VP) [论元数量可变性]
  - Example (1-2) & problematic "semantics" (3-4):

(1) John buttered the toast.

(2) John buttered the toast with the knife in the kitchen.

(3) $butter(j, toast)$

(4) $butter(j, toast, with\_knife, in\_kitchen)$

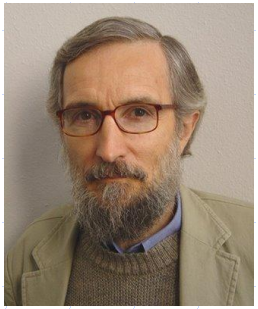
  - butter's arity is not fixed!? ➔ Does it exist? ➔ No ➔ VP prob!
- Davidson: VP resolved (indirectly) by event v (entity for action)!

(5) $\exists v.\ butter(j, toast, v)$

(6) $\exists v.\ butter(j, toast, v) \wedge with\_knife(v) \wedge in\_kitchen(v)$

  - Problems: ontological commitment of events (cf, Quine 1969)
  - Q: can we solve VP without events? A: yes, in type theory.

Type-Theoretical Analysis (I) – Background



- Modern Type Theories (MTTs)
  - Martin-Löf introduced dependent types etc.
- Logic(s) in type theory
  - Curry-Howard principle of propositions-as-types
  - Example logics: PaT (MLTT), Prop (UTT), h-logic (HoTT)
- Relationship between logic and set/type theory

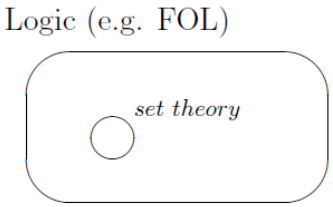


Figure 1: Set theory – a theory in a logic

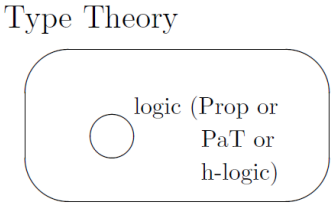


Figure 2: Logic is a part of type theory

- Type-theoretical mechanisms to manipulate logical expressions.
- So, type theory provides a setting for a natural solution to VP!

Type-Theoretical Analysis (II) – Solution to VP

- Recall the VP problem:

  (3) $butter(j, toast)$

  (4) $butter(j, toast, with\_knife, in\_kitchen)$

  Can we have such a "butter"? (Not in traditional logics – VP.)

- In type theory, the answer is yes.

$$butter : \Pi n : N.\ \text{TV-ADV}(n)$$

$$\begin{aligned} butter(0) &: \text{TV-ADV}(0) = \mathbf{e} \to \mathbf{e} \to \mathbf{t} \\ butter(1) &: \text{TV-ADV}(1) = \mathbf{e} \to \mathbf{e} \to \text{ADV} \to \mathbf{t} \\ butter(2) &: \text{TV-ADV}(2) = \mathbf{e} \to \mathbf{e} \to \text{ADV} \to \text{ADV} \to \mathbf{t} \\ &\ldots\ \ldots \end{aligned}$$

- In type theory, we consider
  - Π-types: allowing us to type butter as VP requires.
  - N (type of nats): allowing us to define "butter" and TV-ADV(n) inductively.
  - Part II for details.

What does all this mean?

- Are events necessary? [newly introduced entities for actions]
  - VP has satisfactory solution ➔ dependent typing
  - Other benefits (eg, event talks/perceptual verbs) – doable alternatively
- What does all this mean?
  - Events or dependent typing? [What if Davidson had known dep typing?]
  - MTTs as foundational languages – MTT-semantics
    - A. Ranta. Type-Theoretical Gramma. 1994.
    - S. Chatzikyriakidis & Z. Luo. Formal Semantics in MTTs. Wiley/ISTE, 2020.
    - 罗朝晖. 现代类型论的发展与应用.清华大学出版社，2024年。
  - One may insist on events – MTT-event semantics
    - Dependent event types (Luo & Soloviev 2017)
    - Future work on this?

This page intentionally left blank

Two type constructors in type theory

$$\frac{\Gamma \vdash A\ type \quad \Gamma,\ x{:}A \vdash B\ type}{\Gamma \vdash \Pi x{:}A.B\ type}$$

$$\frac{\Gamma,\ x{:}A \vdash b : B}{\Gamma \vdash \lambda x{:}A.b : \Pi x{:}A.B}$$

$$\frac{\Gamma \vdash f : \Pi x{:}A.B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : [a/x]B}$$

$$\frac{\Gamma,\ x{:}A \vdash b : B \quad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x{:}A.b)(a) = [a/x]b : [a/x]B}$$

- Dependent function types (Π-types)
  - Π-types are typical dependent types.
    - Example: for a function

      g : Πx:Human.Parent(x),

      For any h : Human, g(h) : Parent(h), i.e., g(h) must be h's father/mother.
- Type N of nat numbers allowing inductive definitions:

$$\begin{aligned} \mathcal{E}_N(x, f, 0) &= x \\ \mathcal{E}_N(x, f, succ(n)) &= f(n, \mathcal{E}_N(x, f, n)) \end{aligned}$$

Notes:

- This allows manipulations of logical expressions (logic is internal and "usual" meta-level entities can be manipulated in type theory.)
- This allows inductive definitions of types, eg, TV-ADV next page. (Large elimination or universe – technicality omitted.)

Definition of TV-ADV and butter

- TV-ADV(n) – dependent type of transitive verbs with n adverbial modifiers, with ADV = (e → t) → (e → t):
  - TV-ADV(0) = e → e → t
  - TV-ADV(1) = e → e → ADV → t
  - TV-ADV(2) = e → e → ADV → ADV → t
  - … …
- Example: butter
  - butter : Πn : N. TV-ADV(n)
  - butter(0) = BUTTER : e → e → t
  - butter(n + 1,x,y,$\overline{adv}_{n+1}$) = butter(n, x, y,$\overline{adv}_{n}$) ∧ adv_{n+1}(BUTTER(x), y)

$$
\begin{aligned}
&butter(0) \;:\; \text{TV-ADV}(0) = \mathbf{e} \to \mathbf{e} \to \mathbf{t} \\
&butter(1) \;:\; \text{TV-ADV}(1) = \mathbf{e} \to \mathbf{e} \to \text{ADV} \to \mathbf{t} \\
&butter(2) \;:\; \text{TV-ADV}(2) = \mathbf{e} \to \mathbf{e} \to \text{ADV} \to \text{ADV} \to \mathbf{t} \\
&\ldots\ \ldots
\end{aligned}
$$

Example (VP-form and Conjunctive form)

John buttered the toast.	John buttered the toast with the knife in the kitchen.
butter(0, j, toast)	butter(2, j, toast, with_knife, in_kitchen)
BUTTER(j, toast)	BUTTER(j, toast) ∧ with_knife(BUTTER(j), toast) ∧ in_kitchen(BUTTER(j), toast)

- This (VP-form) is definitionally equal to the conjunctive form.

Summary of benefits

- Solving VP problem (the VP-form)
- Inference as expected:
  - butter(n + 1, ...) ⇒ butter(n, ...)

    For example: butter(1, j, toast, with_knife) ⇒ butter (0, j, toast)
  - Another example——Commutativity of adverbial modifiers by conjunctive form:

    butter(2, j, toast, with_knife, in_kitchen) ⇔ butter(2, j, toast, in_kitchen, with_knife)

Alternatives to some ES benefited phenomena

- Phenomena: event talks, perceptual verbs … …
- Perceptual verbs (see/hear) with tenseless verbs (leave) in clauses:
  - Mary saw John leave.
  - If leave : e → t, then see(m, leave(j)) would be ill-typed.
  - In event semantics, "see" is applied to an event-like entity:

    $\exists v.\ see(v) \wedge ag(v)=m \wedge \exists v'.\ leave(v') \wedge ag(v')=j \wedge pt(v) = v'$
  - Do we have to have events?
- Alternatively, we can have that, in such a case,
  - leave : e → e (i.e., it produces an event-like entity!)
  - Then, see(m, leave(j)) is perfectly well-typed! (Natural solution, we believe.)
- If clauses have adverbial modifications, slightly more sophisticated:
  - Mary saw John leave quickly [meaning postulate; details omitted].

Proofs in Rocq/Coq (proof assistant in type theory)

- **A proof assistant** is an interactive system for constructing and checking formal proofs, such as **Coq**, **Lean**, **Isabelle**, and **Agda**.
- **This Coq code** models transitive verbs with adverbial modifiers and proves that removing an adverb preserves expected semantic entailment.

```
(* Basic imports *)
Require Import Coq.Init.Datatypes.
(* e and t *)
Parameter e : Set.
(* t = Prop as in shallow embedding for simplicity *)
Definition t := Prop.
(* Type of adverbial modifiers *)
Definition ADV := (e -> t) -> (e -> t).
(* Basic semantics of butter without adverbial modification *)
Parameter BUTTER : e -> e -> t.
(* Dependent type of transitive verbs with n adverbial modifiers *)
Fixpoint TV_ADV (n : nat) : Type :=
match n with
| 0 => t
| S m => ADV -> TV_ADV m
end.
(* "Extended conjunction" -- a helper function
*)
(* AND a b : TV_ADV n, where a : TV_ADV n and b : t *)
Fixpoint AND (n : nat) : TV_ADV n -> t -> TV_ADV n :=
match n with
| 0 => fun (a : TV_ADV 0) (b : t) => and a b
| S m => fun a b => fun adv => AND m (a adv) b
end.
(* Inductive definition of butter with base case BUTTER *)
Fixpoint butter (n : nat) (x y : e) : TV_ADV n :=
match n with
| 0 => BUTTER x y
| S m => fun (adv : ADV) =>
AND m (butter m x y) (adv (BUTTER x) y)
end.
(* "Extended implication" -- a helper function *)
(* IMPLIES n a b : Type with a, b : TV_ADV n
*)
```

```
Inductive IMPLIES : forall n, TV_ADV n -> TV_ADV n -> Type :=
| implies_zero :
forall (a b : TV_ADV 0), (a -> b) -> IMPLIES 0 a b
| implies_succ :
forall n (a b : TV_ADV (S n)),
(forall adv : ADV, IMPLIES n (a adv) (b adv)) ->
IMPLIES (S n) a b.
(*
Lemma: AND always entails its first conjunct. *)
Lemma AND_implies_first :
forall n (X : TV_ADV n) (B : t),
IMPLIES n (AND n X B) X.
Proof.
induction n.
- intros X B. simpl. apply implies_zero. intros [H _]. exact H.
- intros X B. simpl. apply implies_succ. intros adv. apply IHn.
Qed.
(* Theorem: dropping an adverbial argument preserves entailment. *)
Theorem butter_Sn_implies_butter_n :
forall (n : nat) (adv : ADV) (x y : e),
IMPLIES n ((butter (S n) x y) adv) (butter n x y).
Proof.
intros n adv x y.
apply AND_implies_first.
Qed.
```

Ready