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

Luo, Z.[1] and Shi, Y.[2,1]

[1]Dept of Computer Science, Royal Holloway, Univ of London
[2]School of Marxism, West Anhui University

Abstract

Davidsonian event semantics [14] is widely accepted as a powerful framework for formal semantics. It has brought about many benefits in semantic construction, which may be summarised into two categories: one is to provide a satisfactory solution to a seemingly intractable problem of variable polyadicity, and the other consists of those benefits that come from the availability of the entities called events that correspond to verb actions.

This paper provides an analysis of event semantics from a general viewpoint of dependent type theory. First, it is shown that the problem of variable polyadicity can be solved by means of dependent typing, without the employment of events. To do this, we only extend the simple type theory with two type constructors and the resulting semantic definitions not only allow variable polyadicity as desired, but obtain logical inferences as expected. We shall discuss why the solution is natural from a type-theoretical point of view (as compared with that in set theory).

We then discuss that most (if not all) of the other benefits of event semantics may already be obtained by alternative means without introducing events as ontological entities. To this end, we consider the evidence for event semantics discussed by Parsons [37], focusing on two particular aspects: event talks and perception words, showing that the former is mostly concerned with timing, and the latter can be dealt with as a special case without introducing events in general.

*This work is partially supported by the EU research network EuroProofNet (COST CA20111) and the China Scholarship Council (Dr Shi, the corresponding author, participated in this work mainly when he was visiting RHUL.) It is also an interim achievement of the Anhui Provincial Philosophy and Social Sciences Planning Project AHSKY2022D229.

1 Introduction

Event semantics originates from Davidson's work in the 1960s [14] to solve the problem of variable polyadicity in modelling adverbial modification, and was further developed in the neo-Davidsonian turn (see Parsons' work [37] among others). Since its emergence more than half a century ago, it has become very popular and widely accepted as a powerful approach to formal semantics.

The benefits of event semantics may be summarised into two categories: one is to provide a satisfactory solution to a seemingly intractable problem of variable polyadicity, and the other consists of those benefits that come from the availability of the entities called events that correspond to verb actions. In this paper, we conduct an analysis of the issues and problems in these two categories, whose solutions have benefited from event semantics, and show that dependent typing presents a new perspective in their studies.

In this section, we first describe the problem of variable polyadicity and then explain how event semantics solves it. Then, in the third subsection, we sketch our main contributions and give a summary of the paper.

1.1 Variable polyadicity

The original motivation for event semantics came from the study of the problem of variable polyadicity [14, 27]. Davidson [14] points out that due to the problem, traditional logical semantics of adverbial modification is unsatisfactory, and introducing the notion of event is a good way to solve the problem.

The variable polyadicity problem can be illustrated by means of the following example sentences (1) and (2), where the latter has got two more adverbial modification phrases:

(1) John buttered the toast.

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

It is clear that there can be indefinitely many adverbial modification phrases in such sentence formations.[1]

A question is: in the above sentences, what is the semantic interpretation of the verb 'butter'? In a tentative attempt, one might consider that it is a predicate that takes the subject and the object ('John' and 'the toast')

[1]Note that the adverbial modifiers we are concerned with in this paper are only those that Bennett [3] calls 'standard' ones, excluding the negative or frequency adverbs such as 'never' or 'seldom' and the non-commital adverbs such as 'allegedly'.

and the adverbial modifiers ('with the knife' and 'in the kitchen') as its arguments, having the following two as semantics of (1) and (2), respectively:

(3) $butter(j, toast)$

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

where j and $toast$ of type $\mathbf{e}$ are the semantics of 'John' and 'the toast', and $with\_knife$ and $in\_kitchen$ of type ADV are the semantics of adverbial phrases 'with the knife' and 'in the kitchen', respectively. (Here, it is not important what ADV is, but see below.) However, there is a problem: what is the type of *butter* in (3) and (4)? For those terms to be well-typed, *butter* in (3) and (4) would need to have the following two different types, respectively:

(5) $\mathbf{e} \to \mathbf{e} \to \mathbf{t}$

(6) $ADV \to ADV \to \mathbf{e} \to \mathbf{e} \to \mathbf{t}$

But this means that the word 'butter' does not seem to be able to have a single interpretation (rather unintuitively!) – its interpretations in (1) and (2) are different terms with different types (5) and (6). In other words, the common semantic term *butter* does not exist. This is the variable polyadicity problem as explained by Davidson (and Kenny). Note that, in the above example, we have only considered adding two more adverbial modifiers to obtain (2) from (1). The intuitive understanding that one may have arbitrarily many such adverbial modifiers seems to make the situation worse.

Let us be clear: the above problem of variable polyadicity arises because, in usual logical systems like first-order logic or simple type theory (higher-order logic), when a predicate symbol is introduced, the number of arguments the predicate should take to form a logical proposition is fixed. There is no such a predicate that could take arbitrarily many arguments to form propositions. So, in the above example, the predicate *butter* in (3) and (4) does not exist in ordinary logical systems and, therefore, the semantic interpretations (3) and (4) are problematic.

It is worth pointing out that, besides the variable polyadicity problem, there are obvious logical relationships between such action sentences as exemplified above, which, however, may not be easy to obtain in traditional logical semantics. For example, intuitively, the semantics of (2) should logically imply that of (1). Even if the semantics (3) and (4) were allowed, would we have that (4) implies (3)? If not, what should one do? In such cases, one would usually have to resort to *ad hoc* meaning postulates to

enforce such relations and this is usually cumbersome and would better be avoided. [2]

1.2 Davidsonian event semantics

Davidson [14] introduced the notion of event, pioneering event semantics, whose modeling method can be summarized as follows:

1. Davidson believes that action verbs semantically introduce (hidden) event variables quantified by existential quantifiers.

2. The semantics of verbs and adverbs (or adverbial phrases) can be described as predicates whose domain consists of all events.

3. Researchers in the neo-Davidsonian period introduced semantic descriptions with thematic roles[3], thereby further simplifying the modeling method.

The neo-Davidsonian event semantics of (1) and (2) can be expressed as (9) and (10), respectively, where the functions ag and pt from events to entities describe the thematic roles called agent and patient, respectively,

[2] In Montague semantics, besides the approach taken in (3) and (4), there are other ways to interpret the sentences like (1) and (2), but they would also require meaning postulates to obtain expected logical relationships. For instance, in simple type theory (higher-order logic), one may consider the following semantic types for the relevant words and phrases:

$$\begin{aligned} butter &: \mathbf{e} \to \mathbf{e} \to \mathbf{t} \\ with\_knife &: (\mathbf{e} \to \mathbf{t}) \to \mathbf{e} \to \mathbf{t} \\ in\_kitchen &: (\mathbf{e} \to \mathbf{t}) \to \mathbf{e} \to \mathbf{t} \end{aligned}$$

Then, (1) and (2) can be interpreted as (7) and (8), respectively:

(7) $butter(j, toast)$

(8) $in\_kitchen(with\_knife(butter(j)))(toast)$

However, in order to have that (8) implies (7), we would have to impose meaning postulates to enforce it: for instance, the meaning postulates could be that both $with\_knife(p, x) \supset p(x)$ and $in\_kitchen(p, x) \supset p(x)$ are true.

BTW, one may also consider the approach to veridicality in modern type theories [8], although its foundational type theories go beyond Montague's approach.

[3] Thematic role is a linguistic term, also known as thematic relation, which refers to the role played by a noun phrase in an event described by a governing verb. For example, in the sentence 'John ate an apple', 'John' is the doer of the eating event and plays the thematic role called 'agent', while 'an apple' is the object of the event and plays the thematic role called 'patient'.

while *butter*, $with\_knife$, and $in\_kitchen$ are all expressed as predicates with events as their domain that variable v ranges over: [4]

(9) $\exists v.\ butter(v)\ \wedge\ ag(v) = j\ \wedge\ pt(v) = toast$

(10) $\exists v.\ butter(v)\ \wedge\ ag(v) = j\ \wedge\ pt(v) = toast$
$\wedge\ with\_knife(v)\ \wedge\ in\_kitchen(v)$

The event semantics (9) and (10) have obvious advantages over (7) and (8) and are more satisfactory. In particular, the adverbial modifiers are no longer arguments of a verb[5] (actually, in the neo-Davidsonian move, the subject or the object is not, either): the semantics of a verb, say *butter*, now always takes an event as its argument and, as a consequence, the variable polyadicity problem disappears. Furthermore, concerning logical relationships, we now obviously have (10) implying (9) (the former has more conjuncts), and do not have to assume extra meaning postulates.

1.3 A type-theoretical solution and summary of the paper

The above variable polyadicity problem has become the key justification for event semantics (there are also other benefits – see below and §3). People just do not see how to resolve it in traditional logical systems, mainly because it would require extending the base logic on which the whole foundational theory (set theory) is based, so that a predicate could take indefinitely many arguments. This is something unnatural to do, to say the least.[6]

[4] People usually regard events as special entities of type **e** and think that events do not form a distinguished type (see [45] for a discussion of the 'event modification problem'). The author believes that it is better to introduce a type *Event* of all events and the event quantifications would become '$\exists v : Event$'. As for whether an event is an entity (that is, whether *Event* is a subtype of **e**), it can be left open if it is not important. In this paper, this matter is not important.

[5] It is worth noting that adverbial modifiers of a verb are taken to be arguments only in the approach illustrated by Davidson [14] (and Kenny [27]), which of course has motivated event semantics. In most of the studies afterwards, people have 'modifiers as modifiers', as a reviewer correctly points out.

[6] There are several papers concerned with proposing new logical systems in order to deal with the variable polyadicity problem. These include the work by Graves [21] and that by Grandy [20], among others. However, because the theoretical foundation is set theory, which is based on logic, a change of the basic logical system has not been popular and, partly because of this, these systems have not become influential. (See §4.1 for further comments.) There are also other problems: some of such systems (for example, [21]) are proposed mainly to fix the variable polyadicity problem, even involving the linguistic notion of thematic roles in its formulation. People who are not doing logical semantics would certainly question the suitability of such a system as a foundational logic.

However, things are different from a type-theoretical point of view. Unlike set theory, type theory is not a theory in traditional logical systems such as first-order logic. Instead, logic is only a part of type theory and, in a modern type theory, there are many non-logical constructs which can provide flexible manipulation mechanisms of logical expressions. (See §4.1 for more technical explanations in this respect.) In this paper, we will demonstrate that, extending Church's simple type theory $\mathcal{C}$ [11], as employed by Montague in formal semantics [35], by two type constructors (dependent function types and a type of natural numbers), will result in a language, called $\mathcal{C}^+$, where a predicate (and a function in general) may be parameterised by numbers to indicate how many arguments it may take to form logical propositions. This naturally solves the variable polyadicity problem: we can have a function *butter* that, when applied to a number, results in a predicate $butter(n)$ that takes n further arguments to form a proposition.[7]

It is worth pointing out that the system $\mathcal{C}^+$ is actually a (small) subsystem of UTT [29], an (impredicative) modern type theory employed in doing formal semantics in modern type theories (MTT-semantics) [8, 31]. Compared with Davidsonian event semantics, our proposed solution offers significant advantages in both formal and ontological economy. By employing a minimal type-theoretical extension of the underlying logical system without introducing events as additional ontological commitments, we aim to achieve explanatory adequacy while maintaining greater simplicity and conceptual clarity. (See §4 for more discussions.)

The introduction of events to formal semantics has brought in both benefits and potential issues or even problems. Besides solving the variable polyadicity problem, event semantics has many other benefits in semantic constructions, from small issues such as commutativity of semantic interpretations of adverbial modifiers, to larger ones such as linking tense and aspect to events in semantics. As discussed in the paper, it seems that most (if not all) of these benefits, except the solution to the variable polyadicity problem, can be obtained by alternative means without introducing events as ontological entities in the semantic framework (see §3). Together with our proposed solution to the variable polyadicity problem in $\mathcal{C}^+$, this seems to suggest that the introduction of event in event semantics may not be necessary as many people would have believed in. We contend that it is important for people to realise this, at least in principle.

The introduction of the notion of event is not without controversy. Ex-

[7] Our formal definition in §2 is slightlly different: there, $butter(n)$ takes $n + 2$ more arguments to form a proposition, with n only indicating the number of adverbial modifiers.

amples of such include, to mention a few: (1) People understand that the employment of events introduces extra ontological commitments that may not be justified easily. (2) It seems very difficult, if possible, to determine the individuation criteria for events (put in another way, it is unclear what reasonable criteria of identity for events are). (3) The event quantifiers in event semantics may have unexpected interferance with other quantifiers in semantic interpretations, leading to questionable or even incorrect semantics. Such considerations have further mystified the notion of event. We contend that the semantics community would be better off if we realise that events are not in principle necessary for formal semantics. Of course, it may be beneficial in practice in semantic construction.

In the following §2, we describe and study the proposed type-theoretical solution, first illustrating the role that dependent typing plays in the proposal (§2.1) and then describe the formal extension of simple type theory and how the new constructs facilitate the implementation of the proposal (§2.2). Then, in §3, we shall consider the benefits of event semantics in the second category, i.e., those because of the presence of events as entities for verb actions. We shall focus on event talks and perception words in §3.1 and §3.2, respectively. §4 discusses two issues: why the proposal to solve the variable polyadicity problem is natural and why we think that the extension is non-ontological. The final section discusses briefly some related and future work.

2 A Type-Theoretic Solution to Variable Polyadicity

In this section, a type-theoretical solution to the variable polyadicity problem is described. We shall first explain what a dependent type is and illustrate informally how the construction works, before getting into the more formal definitions. After describing the proposed solution, we shall discuss several relevant aspects and make some remarks.

2.1 Dependent typing

In this paper, we shall extend the simple type theory with dependent function types (Π-types) and a type N of natural numbers (see §2.2.1 below for a more detailed description).

To give semantic interpretations to action verbs as exemplified in the variable polyadicity problem, we shall employ dependent types TV-ADV(n)

and IV-ADV(n) to help us describe the types of adverbial modification phrases for transitive verbs like 'butter' and intransitive verbs like 'talk', respectively. For specific numbers, we have:

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

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

where $\text{ADV} = (\mathbf{e} \to \mathbf{t}) \to (\mathbf{e} \to \mathbf{t})$ is the type of adverbial modification phrases. For instance, for transtive verbs like *butter*, their semantic types are the following Π-type:

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

Any object f of this type will first take a number argument n, then $f(n)$ is of type $\text{ADV} \to ... \to \text{ADV} \to \mathbf{e} \to \mathbf{e} \to \mathbf{t}$, where ADV occurs n times. If f is *butter*, we then have:

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

This is exactly what we have hoped for. More formal definitions will be given below.

2.2 Proposed solution to the variable polyadicity problem

In this section, we describe the formal mechanisms that extend Church's simple type theory in order to define type-valued functions such as TV-ADV. We shall first describe the system $\mathcal{C}^+$, an extension of Church's simple type theory $\mathcal{C}$ [11], as employed in Montague's semantics [35], with dependent

function types and a type of natural numbers, and then the proposed solution to the variable polyadicity problem.[8]

2.2.1 The underlying type theory $\mathcal{C}^+$

Let $\mathcal{C}$ be Church's simple type theory [11] as employed by Montague for formal semantics [35].[9] The type system $\mathcal{C}^+$ extends $\mathcal{C}$ with the type N of natural numbers and dependent function types (Π-types). The inference rules for $\mathcal{C}^+$ consist of those for $\mathcal{C}$ and those for type constructors Π and N, all listed in Appendix A. We now give brief explanations of Π-types and N.

Dependent function types (Π-types). The dependent function type (Π-type) is a typical form of dependent type. If A is a type, and $B[x]$ is also a type, where $B[x]$ can depend on an object x of type A, then $\Pi x{:}A.B[x]$ is a type. An object of $\Pi x{:}A.B[x]$ is a function f that satisfies the following condition: for any $a : A$, the type of $f(a)$ is $B[a]$. Note that the type $B[a]$ of $f(a)$, the result of applying f to a, depends on the input a – that is why the Π-type is a dependent type.

Here is a very simple example: assume that $Human$ is the type of human beings and, for any $x : Human$, $Parent(x)$ is the type of parents of x, then the Π-type $\Pi x{:}Human.Parent(x)$ contains the functions f as its objects: for any $h : Human$, the type of $f(h)$ is $Parent(h)$; that is, $f(h)$ is either the father or the mother of h.

The formal inference rules for the Π-types can be found in Appendix A.2. Note that a (non-dependent) function type is just a special form of a Π-type: $A \to B$ is just a notation for $\Pi x{:}A.B$ when x does not occur free in B.

The type N of natural numbers. Besides the usual introduction, elimination and definition rules (see Appendix A.3), N also has a large elimination rule:

$$\frac{\Gamma \vdash n : N \quad \Gamma \vdash C\ type \quad \Gamma,\ x{:}N,\ Y\ type \vdash f(x, Y)\ type}{\Gamma \vdash \mathcal{E}_N(C, f, n)\ type}$$

[8] Note that we shall describe our proposal based on Church's simple type theory as employed in Mantague's semantics, partly because we want to resort to the familiarity of many researchers of the simple type theory for easier understanding. Please note that the proposal can also be carried out in a modern type theory – see §4.1.

[9] In most of the formal semantics literature, Church's simple type theory is described model-theoretically. It can also be described proof-theoretically – see, for example, Appendix 1 of [8] or Appendix D of [31] for the proof-theoretic description of $\mathcal{C}$ in the context of studying formal semantics. (Also see the Appendix A.1 of this paper.)

The above rule states that the elimination operator $\mathcal{E}_N$ is also large so that it allows one to define type-valued functions with N as their domain – see the equality laws in Appendix A.3, according to which one can use induction on natural numbers to define functions (and, in particular, type-valued functions by means of large elimination). These include functions like TV-ADV that is used to form dependent types TV-ADV(n), as informally described above and to be formally defined below.

Large elimination has been studied by many authors (see, for example, Dybjer [18] for a discussion in the context of induction-recursion). An alternative to large elimination is to introduce a type universe that contains (the names of) N and Π-types as its objects. The presence of such a universe also allows one to define type-valued functions such as TV-ADV. That is why, in full modern type theoies such as Martin-Löf's type theory [36] and UTT [29] (see §4.1), we do not need large elimination anymore since they already contain universes. In a way, to allow large elimination for N, we have opted for 'simplicity' without introducing another notion (i.e., type universe). However, it should be noted that, to do it this way, we have not gone beyond ordinary dependent type theory.

2.2.2 Description of the proposal

Let ADV = $(\mathbf{e} \to \mathbf{t}) \to (\mathbf{e} \to \mathbf{t})$ (as above). The (semantics of) transitive verbs such as *butter* have type

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

where TV-ADV is a type-valued function, defined by induction on N as:

(12) $\text{TV-ADV}(0) = \mathbf{e} \to \mathbf{e} \to \mathbf{t}$

(13) $\text{TV-ADV}(n+1) = \text{ADV} \to \text{TV-ADV}(n)$

Similarly, the intransitive verbs such as *talk* have type

(14) $talk : \Pi n : N.\ \text{IV-ADV}(n)$

where IV-ADV is defined as:

(15) $\text{IV-ADV}(0) = \mathbf{e} \to \mathbf{t}$

(16) $\text{IV-ADV}(n+1) = \text{ADV} \to \text{IV-ADV}(n)$

Then, intransitive/transitive verb phrases such as talk/butter can be defined as the following Example 2.1 shows.

Example 2.1 (butter) *Let* BUTTER : $\boldsymbol{e} \to \boldsymbol{e} \to \boldsymbol{t}$ *be the semantic interpretation of 'butter' in a sentence with no adverbial modification phrases such as (1). Then, the semantics of transitive verb 'butter', of type (11), can be defined as follows, where in (18),* $n \in \omega$ *and* $\overline{adv}_k$ *stands for the sequence* $adv_1, ..., adv_k$ *for* $k \geq 0$ *(when* $k = 0$, $\overline{adv}_k$ *is the empty sequence):*

(17) $butter(0) = \text{BUTTER}$ *(or,* $butter(0, x, y) = \text{BUTTER}(x, y)$*)*

(18) $butter(n{+}1, \overline{adv}_{n+1}, x, y) = butter(n, \overline{adv}_n, x, y)$ & $adv_{n+1}(\text{BUTTER}(x), y)$

The sentence (2), repeated here as (19), has semantics (20), where $j,\ toast : \boldsymbol{e}$ *and* $with\_knife,\ in\_kitchen : ADV$ *(as explained in §1.1):*

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

(20) $butter(2, with\_knife, in\_kitchen, j, toast)$

Actually, (20) is equal to the following formula:

(21) $\text{BUTTER}(j, toast)$ & $with\_knife(\text{BUTTER}(j), toast)$ & $in\_kitchen(\text{BUTTER}(j), toast)$

which is the basic semantics $\text{BUTTER}(j, toast)$ *in conjunction with two formulas that interpret the adverbial modification phrases. It is not difficult to see the equality between (20) and (21): it can easily be verified by means of the above definition of butter in (17) and (18), according to which we have:*

(22) $butter(2, with\_knife, in\_kitchen, j, toast)$
$= butter(1, with\_knife, j, toast)$ & $in\_kitchen(\text{BUTTER}(j), toast)$
$= butter(0, j, toast)$ & $with\_knife(\text{BUTTER}(j), toast)$ & $in\_kitchen(\text{BUTTER}(j), toast)$
$= \text{BUTTER}(j, toast)$ & $with\_knife(\text{BUTTER}(j), toast)$ & $in\_kitchen(\text{BUTTER}(j), toast)$

□

What we have shown, as illustrated by the above example for *butter*, is that the extra types in $\mathcal{C}^+$ (Π-types and N) can provide useful means to manipulate logical expressions in semantic construction. In particular, we can represent semantic interpretations so that they can take one of the following two forms:

- VP-form:[10] A semantic interpretation of an action verb can take a 'VP-form' where its semantics takes as arguments the adverbial modifiers as well as the subject/object. For instance, the semantics of 'butter' takes a VP-form in (20) to interpret (19).

[10]Here, 'VP' stands for variable polyadicity, not 'verb phrase'.

- Conjunctive form: The VP-form of a semantic interpretation is the same as (or equal to) a 'conjunctive form' where the base meaning of the action verb, applied to its subject/object, is in conjunction with the semantics of each of the adverbial modifiers. For instance, (21) is the conjunctive form of the semantics of (19), where the base meaning of 'butter' is BUTTER and each adverbial modifier is interpreted as $adv(\text{BUTTER}(j), toast)$ with adv being the semantics of the modifier.

For sentences with adverbial modifications, the VP-forms allow us to represent their semantic interpretations as formulas in terms of variable polyadicity of action verbs (cf., Davidson's variable polyadicity problem – see §1.1), while the conjunctive form, equal to the VP-form, brings about many advantages obtainable from an event-based semantics.

An action verb has a single semantics, although it is parameterised by different numbers n when used in semantic constructions for sentences with n adverbial modifiers. For example, *butter* is the semantics for 'butter' and $butter(n)$ is the predicate to be used for sentences with n adverbial modifiers. For different numbers n, $butter(n)$'s are different predicates – they take different numbers of arguments (in particular, n adverbial modifiers) to form logical propositions. The relationship between $butter(n)$ for different n is as expected. For example, for *butter* we have:[11]

(23) $butter(n+1, \overline{adv}_{n+1}, x, y) \supset butter(n, \overline{adv}_n, x, y)$

where adv_i's are semantics of the adverbial modifiers. Because of the above, we can deduce, for example,

(24) $butter(2, with\_knife, in\_kitchen, j, toast)$
$\supset butter(1, with\_knife, j, toast)$
$\supset butter(0, j, toast)$

Note that because $butter(0, j, toast) = \text{BUTTER}(j, toast)$ by definition, we have that (the VP-form of) the semantics of (2) implies that of (1), as expected.

In general, we have:

- for the semantics TV of any transitive verb phrase such as 'butter',

$$TV(n+1, \overline{adv}_{n+1}, x, y) \supset TV(n, \overline{adv}_n, x, y)$$

- for the semantics IV of any intransitive verb phrase such as 'talk',

$$IV(n+1, \overline{adv}_{n+1}, x) \supset IV(n, \overline{adv}_n, x)$$

[11] Note that we only consider 'standard' adverbial modifiers here (c.f., Fn 1).

Note that, just like event semantics, we have obtained the expected inference relationships between such sentences concerning their adverbial modifiers without resorting to meaning postulates (or events).

Commutativity of adverbial modifiers is an (arguably small) advantage of event semantics (see, for example, Landman's lecture notes [28]), although one might have to impose meaning postulates to achieve this in the Montague semantics (c.f., Footnote 2). For instance, the semantics of the following two sentences should be logically equivalent (their adverbial modifiers are swapped):

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

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

This commutativity is true for our proposed semantic representation because it has the conjunctive form (and because the conjunction operator & is commutative). To explain this in more details, the VP-forms of the semantics of (25) and (26) are (27) and (28), respectively, with the arguments $with\_knife$ and $in\_kitchen$ swapped:

(27) $butter(2, with\_knife, in\_kitchen, j, toast)$

(28) $butter(2, in\_kitchen, with\_knife, j, toast)$

It is not difficult to see that their corresponding conjunctive forms are only different because two of the conjuncts are swapped, and therefore, those two conjunctive forms are logically equivalent. As a consequence, the VP-forms (27) and (28) are logically equivalent, too.

Remark The above proposal has been implemented in the Coq proof development system [13] for the 'butter' example, including the above inference relationship (23) as a theorem – see Appendix B.1. □

2.3 Remarks

Our above proposal is based on the extension $\mathcal{C}^+$ of simple type theory. Two remarks about the above approach are in order and they will be further elaborated in §4.

1. First of all, the approach is natural from a type-theoretical point of view (c.f., §4.1). In fact, $\mathcal{C}^+$ is (essentially) a subsystem of UTT [29], one of the modern type theories [34, 13, 24] implemented in proof assistants with applications such as formalisations of mathematics and

program verification (and also linguistic semantics [8, 31]). One may take this as saying that the proposal is not based on a change of the underlying logic made specifically to solve the variable polyadicity problem (c.f., Footnote 6); instead, it is based on a formal system studied for a much more general purpose.

2. Compared with events in event semantics, the extensions to obtain $\mathcal{C}^+$ from $\mathcal{C}$ is non-ontological and may be called, in Quine's terminology [40], ideological (c.f., §4.2). In other words, we have employed additional mechanisms for manipulating logical expressions. They may also be called syntactical mechanisms and have nothing to do with ontological existence.

3 Other Benefits of Event Semantics

Davidson's original motivation to introduce the notion of event (and event quantification) into formal semantics was to solve the problem of adverbial modification (as we have studied in the above section). During its studies for more than half a century, people have recognised that event semantics has many other benefits as well, including those exemplified by Parsons [37]. We classify the benefits of event semantics into the following two categories:

1. Category 1 singles out the variable polyadicity problem, as studied above – event semantics provides a good solution.

2. Category 2 consists of several benefits of event semantics, all resulting from the introduction of the notion of events, individual-like entities that correspond to actions associated with verbs.

The introduction of events plays such a role that, for every action associated with a verb, one introduces an entity that corresponds to the action, and this entity is called an event, introduced as an (existentially quantified) event variable. According to our analysis, most (if not all) of the benefits in the second category come from the availability of such newly introduced entities that correspond to actions.

We contend that the variable polyadicity problem may be the only one that requires a substantial change like introducing the notion of event in event semantics (but see the above for our proposed solution without events). In this section, we shall attempt to show that, for those benefits of event semantics in the second category, we can already find alternative ways in

the semantic framework to deal with them without introducing event variables as ontological commitments. Of course, one may not exhaust all the possibilities and what we shall do is to analyse what Parsons [37] considered as evidence for event semantics, in addition to the variable polyadicity problem, in two aspects: (1) times in semantics (in §3.1) and (2) special semantics involving perceptual verbs (in §3.2). We shall use these cases to illustrate how semantic constructions for these phenomena can already be done without events.

The notion of event is also naturally related to a proper semantic understanding of nominalisation. In §3.3, we shall discuss some pieces of work that try to study nominalisation without introducing events.

3.1 Event talks and time

The notion of event is closely related to time, as an event is naturally associated with the time at which it occurs. However, there seems to be an important difference: tense and aspect exist in language and, therefore, times already occur in expressions in most (if not all) sentences, while events do not – events only occur in some sentences that refer to them directly. Put in another way, time appears at the surface level in most sentences, whilst events do not. That explains why it is natural for people to consider semantic frameworks with time as one of their key elements, whilst having events as an ontological commitment may be more questionable.

Parsons [37] gave some examples of 'event talks' as evidence for event semantics – the following are such examples given by Parsons, where (29) involves explicit event reference, while (30)'s reference to events is implicit.

(29) After the singing of the Marseillaise, John saluted the flag.

(30) After the Marseillaise was sung, John saluted the flag.

Using event semantics, one can model these sentences as (31), where $after(e', e)$ is the formula expressing that event e' happens after event e.

(31) $\exists e.\ saluting(e)\ \&\ ag(e) = j\ \&\ pt(e) = flag\ \&$
$\exists e'.\ singing(e')\ \&\ pt(e') = M\ \&\ after(e', e)$

If we use t_e to express the time at which event e takes place, then $after(e', e)$ can be equivalently expressed as $t_e < t_{e'}$.

Many of such cases as the above are concerned with times at which events take place, and their semantics can be given in a semantic framework that already has times expressible for dealing with tensed sentences, without

resorting to events. For example, in such a framework, the semantics of (29) or (30) can be given as (32):

(32) $\exists t, t'.\ sing(j, M, t)\ \&\ salute(j, flag, t')\ \&\ t < t'$

where a semantic predicate for an action verb takes one more argument, expressing the time that the action takes place. For example, $sing(j, M)$ becomes $sing(j, M, t)$, and similarly for *salute*.[12]

Another example Parsons considers involves event quantification: for the sentence (33), he gives event semantics (34), where the formula $IN(e, e')$ is undefined:

(33) In every burning, oxygen is consumed.

(34) $\forall e.\ burning(e) \supset \exists e'.\ consuming(e')\ \&\ pt(e') = O_2\ \&\ IN(e, e')$

Such sentences can also be given semantics in a timed framework, without using events. For instance, the above sentence (33) may be given the following semantics (35).

(35) $\forall x \forall t.\ burn(x, t) \supset consume(O_2, t)$

To rephrase (35), in every burning x that takes place at t, oxygen is consumed at t as well. We contend that this also represents the semantics correctly, without using events.

3.2 Perception words

Another case that Parsons [37] puts forward is how to give suitable semantics to a sentence that features a perceptual verb followed by a clause lacking explicit tense, as exemplified by the following sentence.

(36) Mary saw John leave.

Let's assume that the semantic type of 'see' be $\mathbf{e} \to \mathbf{e} \to \mathbf{t}$. The question is what the semantic type of 'leave' should be. A usual semantics for verbs would not work for, if *leave* were of type $\mathbf{e} \to \mathbf{t}$, then (37) would not be well-typed and, hence, cannot be considered as the semantics of (36):

(37) (#) $see(m, leave(j))$

[12] Studies of such 'positional' notations for temporal expressions may be traced back to the work by Łoś in the 1940s (his thesis in 1947 – see, for example, [44]) and one may also consult other systems based on such positional calculi (see, for example, [17]), which are different from Prior-style temporal logics with temporal operators [39]. Also, here, the time t could either be an instance or an interval, and we omit formal details.

One may ask: what if we take *see*'s type to be $\mathbf{e} \to \mathbf{t} \to \mathbf{t}$? Yes, in that case, (37) would be well-typed. However, there is a problem: in that case, (37) would not be suitable as the semantics of (36) because it is closer to the meaning of the following sentence with a that-clause:

(38) Mary saw that John left.

The above (38) has a different meaning from (36): with the that-clause, it means that Mary became aware that the leaving event took place (usually without seeing it directly).

In event semantics, the above problem can be resolved because individual-like entities (events of type $\mathbf{e}$) are introduced for actions associated with verbs and the verbs are all interpreted as predicates over events, of type $\mathbf{e} \to \mathbf{t}$. For example, the event semantics of (36) can be (39):

(39) $\exists e.\ see(e)\ \&\ ag(e) = m\ \&$
$\exists e'.\ leave(e')\ \&\ ag(e') = j\ \&\ pt(e) = e'$

The key is that the entity e' is now available to be the patient of e; that is, e' is the event seen in the event e. The event-based analysis models this by treating the embedded clause as an event perceived by the subject.

We contend that interpretating such sentences involving perceptual verbs does not necessarily require us to use event semantics, where events are introduced in general as ontological commitments for all sentences. Instead, we'd only need to do something in such special cases as interpreting perceptual verbs with tenseless clauses. Note that the introduction of events has essentially turned an action (John's leaving in (36)) into an entity (the event e' in (39)), so that one can use e' in semantic construction. A similar thing can be done without introducing events in general – here is our proposal, which we explain for sentence (36).

We assume that, as usual, the verbs 'see' and 'leave' have semantic types (40) and (41), respectively, where we choose to use LEAVE instead of *leave* because we will use the latter below for the general semantics of 'leave' for which LEAVE becomes a special case:

(40) $see : \mathbf{e} \to \mathbf{e} \to \mathbf{t}$

(41) $\textsc{leave} : \mathbf{e} \to \mathbf{t}$

In order to interpret (36), we consider a special mapping $\mathbf{E}$, of type $\mathbf{t} \to \mathbf{e}$, which turns a logical formula into an entity: for example, $\mathbf{E}(\textsc{leave}(j))$:

$\mathbf{e}$.[13] Then, the sentence (36) can be interpreted as (42):

(42) $see(m, \mathbf{E}(\text{LEAVE}(j)))$

We emphasise that introducing the mapping $\mathbf{E}$ is only for the interpretation of such sentences involving perceptual verbs, not in general. Put in another way, event-like entities are not introduced in general as ontological commitments, but only when they are absolutely needed in special cases (and, in this case, for perceptual verbs).

There is a further issue one may raise: what if one adds adverbial modifiers to the senseless verb 'leave' in (36), as exemplified by the following sentences (43) and (44), where one and two adverbial modifiers are added, respectively:

(43) Mary saw John leave quickly.

(44) Mary saw John leave quickly and quietly.

To interpret such sentences as (36), (43) and (44) in general, we need to consider the general interpretation of verb phrases when a verb (e.g., 'leave') takes arbitrarily many adverbial modifiers (c.f., the definition of *butter* in §2.2.2). For 'leave', its semantics is of the following type (for intransitive verbs, as (14) in §2.2.2):

(45) $leave : \Pi n : N.\ \text{IV-ADV}(n)$

where IV-ADV's definition is given in (15) and (16) in §2.2.2. We can now define the semantics of the 'leave-phrases' as follows:

(46) $leave(0, x) = \text{LEAVE}(x)$

(47) $leave(n+1, \overline{adv}_{n+1}, x) = leave(n, \overline{adv}_n, x)\ \&\ adv_{n+1}(\text{LEAVE}, x)$

The semantics of (36), (43) and (44) can now be given as (48), (49) and (50), respectively.

(48) $see(m, \mathbf{E}(leave(0, j)))$

(49) $see(m, \mathbf{E}(leave(1, quickly, j)))$

[13] Here, $\mathbf{E}$ alludes to 'entity' or 'event'. Instead of assuming the existence of $\mathbf{E}$ directly, one might also consider a more elaborate treatment, introducing $\mathbf{E}$ as follows by going through unit types as a function 'composition': $\mathbf{E}(p) = E_0(p, *(p))$, where $E_0 : \Pi p : \mathbf{t}.\ \mathbf{1}(\mathbf{t}, p) \to \mathbf{e}$, and $* : \Pi p : \mathbf{t}.\ \mathbf{1}(\mathbf{t}, p)$ is the constructor of the unit types. The parameterised unit types $\mathbf{1}(A, a)$ are singleton types with only one object a of type A, whose explanations are omitted (see, for example, [30] among others).

(50) $see(m, \mathbf{E}(leave(2, quickly, quietly, j)))$

In order to obtain the desired inference relationships between sentences with such a structure, we need to impose the following meaning postulate: for *see* and *leave*, we assume (51), where $x : e$, $n : N$ and $adv_i : (\mathbf{e} \to \mathbf{t}) \to (\mathbf{e} \to \mathbf{t})$.

(51) $see(x, \mathbf{E}(leave(n+1, \overline{adv}_{n+1}, x))) \supset see(x, \mathbf{E}(leave(n, \overline{adv}_n, x)))$

Of course, because of (51) and the transitivity of logical implication, we can prove the following (52):

(52) $see(x, \mathbf{E}(leave(n, \overline{adv}_n, x))) \supset see(x, \mathbf{E}(leave(m, \overline{adv}_m, x)))$
for any natural numbers $m \leq n$.

With this, it is easy to see that (50) $\supset$ (49) $\supset$ (48), that is, informally we have: (44) implies (43) implies (36), as expected.

Remark The above has been implemented in the Coq system, including (52) as a theorem – see Appendix B.2. □

Note that the meaning postulate (51) does not cover everything we expect to hold. For example, consider the following sentence (53), where 'and' connects two clauses 'John leave' and 'Kevin run'. Then, intuitively, it would be the case that (53) implies (36) which, strictly speaking, is not covered by (51). However, in this case, one may argue that (54) rephrases (53) and therefore, because (54) implies (36), we are OK.

(53) Mary saw John leave and Kevin run.

(54) Mary saw John leave and Mary saw Kevin run.

3.3 Nominalisation: a discussion

The introduction of an entity to stand for an action has naturally provided an elegant solution to semantic understanding of nominalisation. For example, consider the sentence (55), where both 'talked' and its nominalisation 'the talk' occur. In its event semantics (56), the event variable e is the entity that corresponds to the talking action performed by John.

(55) John talked and the talk was interesting.

(56) $\exists e.\ talking(e)\ \&\ ag(e) = j\ \&\ interesting(e)$

Because of the existence of the entity e, one can then form $interesting(e)$ to say that the talking action is interesting as if it is the event e. In this way, event semantics establishes the connection between action verbs and their nominalisations naturally, which one might even consider as a side-effect when trying to resolve the VP problem.

It is arguable whether event semantics is essential in resolving the nominalisation problem. In the literature, there exist different approaches to deal with nominalisation including, for example, Chierchia's work [9, 10]. In these approaches, event semantics is not used. However, one may also argue that somehow event-like structures are employed in these approaches including, for instance, discourse referents in Discourse Representation Theory [25, 26, 23]. In this paper, we shall not discuss these approaches in detail.

4 Discussions

This section discusses some useful background information and philosophical arguments behind the proposed technical solution to the problem of variable polyadicity (see, in particular, §2). We hope that this could lead to a better understanding of the central theme of the paper.

4.1 Type theory and logic

We argue that our proposed solution to the variable polyadicity problem is natural from a type-theoretical point of view, especially when it is compared with that in the traditional set-theoretical framework.

Modern studies of type theory started from Martin-Löf's work in early 70s on foundations of constructive mathematics (see Martin-Löf [33] and subsequent developments [34, 36]). Type theory and the associated proof assistants have interesting applications in formalisation of mathematics (see, for example, [4]), program verification (see, for example, [38] for a recent development), and foundations of mathematics, including the recent study of univalent foundations [24, 5]. Type theory, and the idea of dependent typing in particular, have been applied to linguistic semantics as well (see [1, 2, 8, 12, 22, 31, 41, 42, 43, 46], among others).

An important, but often neglected, feature of type theory is its relationship with logic. This may be shown by explaining its difference from set theory. Set theory (or a set theory) is a theory in a logical system (e.g., first-order logic FOL), as depicted by the picture on the left in Figure 1. A type theory, however, is not a theory in logic. Instead, a type

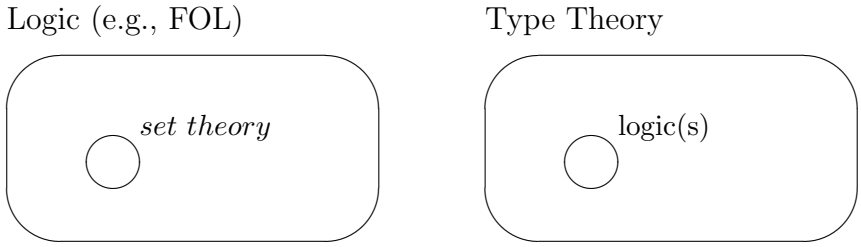

Figure 1: Relationships between logic and set theory/type theory

theory is at the same level as a logical system (e.g., FOL): it is specified by means of inference rules. Importantly, logic is a part of type theory, as depicted by the picture on the right in Figure 1. For instance, one has the propositions-as-types logic in Martin-Löf's type theory [34], a higher-order logic in impredicative type theories such as UTT [29] and the h-logic (with propositional truncation) in homotopy type theory [24].

In particular, in type theory, there exist other non-logial mechanisms, some of which can be used to manipulate logical expressions in type theory. That is why it is natural to think that there can be mechanisms in a formal system that may be used to manipulate logical expressions. The extensions of Π-types and N in $\mathcal{C}^+$, as described in §2.2.1, are such mechanisms. In fact, $\mathcal{C}^+$ is a subsystem of the impredicative type theory UTT [29] (or, with a small qualification, the Coq's type theory pCIC [13]).[14] Therefore, we believe that our solution to the variable polyadicity problem as proposed based on the extension $\mathcal{C}^+$ is natural from a type-theoretical point of view.

4.2 Events: ontological commitments and discussions

Event semantics has been very popular and widely accepted as a powerful mechanism for formal semantics. However, the notion of event is rather mysterious and has not been clearly understood. One would ask, for example: What is an event? Can events be counted and, if so, how? Unfortunately, such basic questions have not received satisfactory answers. An important issue concerning this is that it is unclear what the notion of identity between events should be, and many people believe that the notion of identity

[14]Formally, a detail need be clarified: there are universes in a type theory and, if one has universes, large elimination (as found for N in this paper) becomes unnecessary.

criterion is extremely important, saying, for instance, there are no entities without identity (see, for example, Geach [19]). Without a good understanding of the identity criteria, the notion of event would remain mysterious and a better understanding is called for.

The issue we would like to discuss here, albeit briefly, is about the ontological commitments on events in Davidsonian event semantics.[15] Davidson argued, with the (negative) evidence concerning the variable polyadicity problem, that action verbs carry implicit uses of existentially quantified event variables. However, this may not be the case if events do not necessarily exist, and it may be reasonable to think of events not as ontological entities, but rather just as an intermediate mechanism helpful in semantic constructions.

Quine [40] pointed out that, in theoretical development, there are different kinds of commitments: ontological commitments and idealogical commitments. For example, Davidson regards events as ontological commitments in event semantics, while our extension of simple type theory to form $\mathcal{C}^+$ should be considered idealogical in the sense that they only provide syntactical tools for semantic construction. As another example, if we introduce time in a semantic theory, it is usually regarded as an ontological commitment. Of course, as we commented on in §3.1, time appears at the surface level in most sentences, whilst events do not. Therefore, it is natural for time to be considered as one key component in a semantic framework, while having events as an ontological commitment would be more questionable.

Without considering events as ontological commitments, one may think of the framework of event semantics as, at least theoretically, an intermediate one to provide a useful toolkit in semantic construction. It would be imaginable that one could produce a tool that automatedly transforms a semantics in event semantics into an 'equivalent' one without events. Of course, even so, this may only be a theoretical possibility, and further work is needed to explore this in such a direction.

5 Conclusion

In this paper, we have introduced a type-theoretical approach to addressing the problem of variable polyadicity in formal semantics, a challenge traditionally managed through Davidsonian event semantics. Extending simple

[15] Concerning whether events are considered as ontological entities, philosophers have taken different views, for example, about the nature of the variables under event quantifier (c.f., Bennett's work [3]). We do not make such subtle differences in this paper.

type theory with simple dependent typing mechanisms, the semantic framework accommodates predicates of varying arities without incurring the additional ontological commitments typically associated with event semantics. Our proposed solution to the variable polyadicity problem also supports expected logical inference, which are traditionally attributed to event-based analyses. Also analysed are some of the linguistic phenomena that have been argued to necessitate event semantics, such as event reference and the semantics of perception verbs. Our analysis demonstrates that these phenomena can equally be accounted for without explicitly introducing events as ontological entities.

There are issues about event semantics that are not studied here including, for example, the so-called Event Quantification Problem, the interference problem of the event quantifier with other quantifiers in semantic constructions, as pointed out and studied by several researchers (Champollion, Wintor, and others [6, 45, 16]). The EQP problem may be resolved by introducing so-called Dependent Event Types [32], but we do not discuss it here. Of course, if we do not introduce the event quantifier in the semantic framework, we would not suffer from the EQP problem.

Resolving the problem of variable polyadicity, we have proposed a method to allow arbitrarily many adverbial modifiers as arguments of a verb in logical semantics. What about adjectival modifiers – can we do similar things in a type-theoretic framework? (Thanks to a reviewer for raising this issue.) It seems to us that intuitively the answer is positive, although we have not worked out the technical details. We leave it as future work.

Looking ahead, our work suggests several promising directions for future research, including the exploration of automated methods for translation between event-based and event-free semantic representations and extending type-theoretic analyses to consider more linguistic structures. We hope that this contribution will encourage semanticists to reconsider the ontological assumptions underpinning formal semantic theories and further investigate the potential of dependent type theory in linguistic semantics.

References

[1] N. Asher. *Lexical Meaning in Context: a Web of Words*. Cambridge University Press, 2012.

[2] D. Bekki and K. Mineshima. Context-Passing and Underspecification in Dependent Type Semantics. In S. Chatzikyriakidis and Z. Luo, editors, [7], pp. 11–41. Springer, 2017.

[3] J. Bennett. *Events and Their Names*. Oxford University Press, 1988.

[4] K. Buzzard. What is the point of computers? A question for pure mathematicians. *Companion paper to a talk in International Congress of Mathematicians (ICM 2022)*, 2022.

[5] S. Centrone, D. Kant, and D. Sarikaya, editors. *Reflections on the Foundations of Mathematics: Univalent Foundations, Set Theory and General Thoughts*. Springer, 2019.

[6] L. Champollion. The interaction of compositional semantics and event semantics. *Linguistics and Philosophy*, 38:31–66, 2015.

[7] S. Chatzikyriakidis and Z. Luo, editors. *Modern Perspectives in Type-Theoretical Semantics*. Springer, Cham, 2017.

[8] S. Chatzikyriakidis and Z. Luo. *Formal Semantics in Modern Type Theories*. Wiley/ISTE, 2020.

[9] G. Chierchia. *Topics in the Syntax and Semantics of Infinitives and Gerunds*. PhD thesis, University of Massachusetts, 1984.

[10] G. Chierchia. Formal semantics and the grammar of predication. *Linguistic Inquiry*, (3), 1985.

[11] A. Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5(1), 1940.

[12] R. Cooper. Records and record types in semantic theory. *J. of Logic and Computation*, 15(2), 2005.

[13] The Coq Team. *The Coq Proof Assistant Reference Manual (Version 8.1), INRIA*, 2007.

[14] D. Davidson. The logical form of action sentences. *In: S. Rothstein (ed.). The Logic of Decision and Action. University of Pittsburgh Press*, 1967. (Reprinted in [15]).

[15] D. Davidson. *The Essential Davidson*. Oxford University Press, 2006.

[16] P. de Groote and Y. Winter. A type-logical account of quantification in event semantics. *Logic and Engineering of Natural Language Semantics 11*, 2014.

[17] D. Dowty. *Word Meaning and Montague Grammar*. Springer, 1979.

[18] P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *The Journal of Symbolic Logic*, 65(2), 2000.

[19] P. Geach. *Reference and Generality: An Examination of Some Medieval and Modern Theories*. Cornell University Press, Ithaca, 1962.

[20] R. Grandy. Anadic logic and English. *Synthese*, 32(3-4), 1976.

[21] P. Graves. Argument deletion without events. *Notre Dame Journal of Formal Logic*, 34(4), 1993.

[22] J. Grudzińska and M. Zawadowski. Generalized quantifiers on dependent types: A system for anaphora. In S. Chatzikyriakidis and Z. Luo, editors, [7], pages 95–131. Springer, 2017.

[23] I. Heim. *The Semantics of Definite and Indefinite Noun Phrases*. PhD thesis, University of Massachusetts, 1982.

[24] HoTT. *Homotopy Type Theory: Univalent Foundations of Mathematics*. The Univalent Foundations Program, Institute for Advanced Study, 2013.

[25] H. Kamp. A theory of truth and semantic representation. *In J. Groenendijk et al (eds.) Formal Methods in the Study of Language*, pages 189–222, 1981.

[26] H. Kamp and U. Reyle. *From Discourse to Logic*. Kluwer, 1993.

[27] A. Kenny. *Action, Emotion and Will*. Routledge and Kegan Paul, 1963.

[28] F. Landman. *Events and Plurality: The Jerusalem Lectures*. Studies in Linguistics and Philosophy, vol 76. Springer, 2000.

[29] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*. Oxford University Press, 1994.

[30] Z. Luo. Manifest fields and module mechanisms in intensional type theory. In S. Berardi, F. Damiani, and U. de'Liguoro, editors, *Types for Proofs and Programs, Proc. of Inter. Conf. of TYPES'08, LNCS 5497.*, 2009.

[31] Z. Luo. *Modern Type Theories: Their Development and Applications*. Tsinghua University Press, 2024. (In Chinese. A book in English based on this book is to appear soon).

[32] Z. Luo and S. Soloviev. Dependent event types. In J. Kennedy and R. de Queiroz, editors, *Logic, Language, Information, and Computation. WoLLIC 2017, LNCS*, volume 10388. Springer, 2017.

[33] P. Martin-Löf. An intuitionistic theory of types: predicative part. In H.Rose and J.C.Shepherdson, editors, *Logic Colloquium'73*, 1975.

[34] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[35] R. Montague. *Formal Philosophy*. Yale University Press, 1974. Collected papers edited by R. Thomason.

[36] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: An Introduction*. Oxford University Press, 1990.

[37] T. Parsons. *Events in the Semantics of English*. MIT Press, 1990.

[38] B. Pierce et al. *Logical Foundations (Software Foundations series, Volume 1)*. Electronic textbook, 2018.

[39] A. Prior. *Time and Modality*. Oxford University Press, 1957.

[40] W. Quine. Ontology and ideology. *Philosophical Studies*, 2(1), 1951.

[41] A. Ranta. *Type-Theoretical Grammar*. Oxford University Press, 1994.

[42] C. Retoré. The montagovian generative lexicon ty_n: a type theoretical framework for natural language semantics. *Proc of the 19th International Conference on Types for Proofs and Programs (TYPES 2013), LIPIcs(26)*, 2014.

[43] G. Sundholm. Constructive generalized quantifiers. *Synthese*, 79(1):1–12, 1989.

[44] M. Tkaczyk and T. Jarmużżek. Jerzy Łoś positional calculus and the origin of temporal logic. *Logic and Logical Philosophy*, 28, 2019.

[45] Y. Winter and J. Zwarts. Event semantics and abstract categorial grammar. *Proc. of Mathematics of Language 12, LNCS 6878*, 2011.

[46] M. Zawadowski and J. Grudzińska. Polyadic quantifiers on dependent types. In *Proceedings of the 30th International Workshop on Logic, Language, Information, and Computation (WoLLIC 2024), Switzerland*, 2024.

A Inference Rules for $\mathcal{C}^+$

The inference rules for $\mathcal{C}^+$ consist of those for Church's simple type theory $\mathcal{C}$ and those for N and Π-types.

A.1 Inference rules for $\mathcal{C}$

The following inference rules for $\mathcal{C}$ are based on the description of Church's simple type theory in [32], where in the second rule, $FV(\Gamma)$ is the set of variables that occur freely in Γ.[16]

Rules about contexts, base types and λ-calculus

$$\frac{}{\langle\rangle\ valid} \qquad \frac{\Gamma \vdash A\ type \quad x \notin FV(\Gamma)}{\Gamma,\ x{:}A\ valid} \qquad \frac{\Gamma \vdash P : \mathbf{t}}{\Gamma,\ P\ true\ valid}$$

$$\frac{\Gamma\ valid}{\Gamma \vdash \mathbf{e}\ type} \qquad \frac{\Gamma\ valid}{\Gamma \vdash \mathbf{t}\ type} \qquad \frac{\Gamma,\ x{:}A,\ \Gamma'\ valid}{\Gamma,\ x{:}A,\ \Gamma' \vdash x : A} \qquad \frac{\Gamma,\ P\ true,\ \Gamma'\ valid}{\Gamma,\ P\ true,\ \Gamma' \vdash P\ true}$$

$$\frac{\Gamma \vdash A\ type \quad \Gamma \vdash B\ type}{\Gamma \vdash A \to B\ type} \qquad \frac{\Gamma, x{:}A \vdash b : B}{\Gamma \vdash \lambda x{:}A.b : A \to B} \qquad \frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

Rules about formation and inference of logical formulas

$$\frac{\Gamma \vdash P : \mathbf{t} \quad \Gamma \vdash Q : \mathbf{t}}{\Gamma \vdash P \supset Q : \mathbf{t}} \qquad \frac{\Gamma,\ P\ true \vdash Q\ true}{\Gamma \vdash P \supset Q\ true} \qquad \frac{\Gamma \vdash P \supset Q\ true \quad \Gamma \vdash P\ true}{\Gamma \vdash Q\ true}$$

$$\frac{\Gamma \vdash A\ type \quad \Gamma, x{:}A \vdash P : \mathbf{t}}{\Gamma \vdash \forall x{:}A.P : \mathbf{t}} \qquad \frac{\Gamma, x{:}A \vdash P\ true}{\Gamma \vdash \forall x{:}A.P\ true} \qquad \frac{\Gamma \vdash \forall x{:}A.P(x)\ true \quad \Gamma \vdash a : A}{\Gamma \vdash P(a)\ true}$$

Transformation rule for logical formulas, where $\simeq_\beta$ is the β-conversion.[17]

$$\frac{\Gamma \vdash P\ true \quad \Gamma \vdash Q : \mathbf{t}}{\Gamma \vdash Q\ true} \quad (P \simeq_\beta Q)$$

[16] In the simple type theory $\mathcal{C}$, $FV(\Gamma)$ can be defined as follows: (1) $FV(\langle\rangle) = \emptyset$; (2) $FV(\Gamma, x{:}A) = FV(\Gamma) \cup \{x\}$; (3) $FV(\Gamma, P\ true) = FV(\Gamma)$. Note that when we use $FV(\Gamma)$, Γ is always a valid context (i.e.: $\Gamma\ valid$). Therefore, $FV(\Gamma)$ defined in this way is indeed the set of variables that occur freely in Γ.

[17] As in λ-calculus, β-conversion holds: the term $(\lambda x{:}A.b[x])(a)$ and $b[a]$ are computationally equal (the former computes to the latter) and β-conversion is the equivalence relation induced by this basic computation.

A.2 Inference rules for Π-types

The following inference rules for Π-types are standard ones in a dependent type theory (see, for example, Martin-Löf [34]).

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

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

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

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

A.3 Inference rules for N

Usual rules for type of natural numbers (see, for example, Martin-Löf [34])

$$\frac{\Gamma\ valid}{\Gamma \vdash N\ type} \qquad \frac{\Gamma\ valid}{\Gamma \vdash 0 : N} \qquad \frac{\Gamma \vdash n : N}{\Gamma \vdash succ(n) : N}$$

$$\frac{\begin{array}{l}\Gamma, z{:}N \vdash C(z)\ type \quad \Gamma \vdash n : N \\ \Gamma \vdash c : C(0) \quad \Gamma, x{:}N, y{:}C(x) \vdash f(x,y) : C(succ(x))\end{array}}{\Gamma \vdash \mathcal{E}_N(c,f,n) : C(n)}$$

Large elimination rule for N

$$\frac{\Gamma \vdash n : N \quad \Gamma \vdash C\ type \quad \Gamma,\ x{:}N,\ Y\ type \vdash f(x,Y)\ type}{\Gamma \vdash \mathcal{E}_N(C,f,n)\ type}$$

Definition rules for elimination operator $\mathcal{E}_N$ (simplified as equality laws)

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

where x can either be an object of some type (in the case of ordinary elimination) or a type itself (in the case of large elimination).

B Coq Implementations

B.1 Proposal for Variable Polyadicity

```
(* Basic imports *)
Require Import Coq.Init.Datatypes.
Require Import Coq.Arith.Arith.

(* e and t *)
Parameter e : Set.          (* Type for individuals/entities *)
Definition t := Prop.       (* Type for truth values / propositions *)

(* Adverbial modifier type *)
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 => e -> e -> 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 -> (e -> e -> t) -> TV_ADV n :=
  match n with
  | 0 => fun a b => fun x y => a x y /\ b x y
  | 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) : TV_ADV n :=
  match n with
  | 0 => BUTTER
  | S m => fun (adv : ADV) =>
      AND m (butter m) (fun 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), (forall x y, a x y -> b x y) -> 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.

(* AND always entails its first conjunct, at any adverbial depth. *)
Lemma AND_implies_first :
  forall n (X : TV_ADV n) (B : e -> e -> t),
    IMPLIES n (AND n X B) X.
Proof.
  induction n.
  - intros X B. simpl. apply implies_zero. intros x y [H _]. exact H.
  - intros X B. simpl. apply implies_succ. intros adv. apply IHn.
Qed.

(* Theorem: dropping an adverb layer preserves entailment *)
Theorem butter_Sn_implies_butter_n :
  forall (n : nat) (adv : ADV),
    IMPLIES n (butter (S n) adv) (butter n).
Proof.
  intros n adv.
  apply AND_implies_first.
Qed.
```

B.2 Semantics for Perceptual Verbs

```
(* Basic imports *)
Require Import Coq.Init.Datatypes.
Require Import Coq.Arith.Arith.
Require Import Lia.

(* Entity type and semantic primitives *)
Parameter e : Set.          (* Entity type *)
Definition t := Prop.       (* Propositional type *)
```

```
(* Dependent unit type and event lifting *)
Inductive Unit (p : t) : Set :=
  | ast : Unit p.
Parameter E0 : forall p:t, Unit p -> e.
Definition E (p:t) : e := E0 p (ast p).

(* Adverbial modifier type *)
Definition ADV := (e -> t) -> (e -> t).

(* lexical terms for "Mary saw John leave" *)
Parameter LEAVE : e -> t.
Parameter see : e -> e -> t.
Parameter john mary : e.

(* Dependent type of intransitive verbs with n adverbial modifiers *)
Fixpoint IV_ADV (n : nat) : Type :=
  match n with
  | 0 => e -> t
  | S m => ADV -> IV_ADV m
  end.

(* Inductive definition of leave with base case LEAVE *)
Fixpoint leave (n : nat) : IV_ADV n :=
  match n with
  | 0 => LEAVE
  | S m => fun adv => leave m
  end.

(* Recursively apply adverbs to IV_ADV n -- a helper function *)
Fixpoint apply_advs
  (n : nat) (f : IV_ADV n) (advs : nat -> ADV) (y : e) : t :=
  match n as n' return IV_ADV n' -> (nat -> ADV) -> e -> t with
  | 0 => fun f0 _ y0 => f0 y0
  | S m => fun f1 advs1 y1 =>
      apply_advs m (f1 (advs1 0)) (fun i => advs1 (S i)) y1
  end f advs y.

(* Specialized "adverb-stripping" postulate for leave *)
Axiom MP_leave_layered :
```

```
  forall (p : e -> e -> t)
         (n : nat)
         (x : e)
         (advs : nat -> ADV)
         (y : e),
    p x (E (apply_advs (n + 1) (leave (n + 1)) advs y)) ->
    p x (E (apply_advs n (leave n) advs y)).


(* Multi-layer adverb-stripping for leave: removing m layers *)
Theorem MP_leave_multi :
  forall (p : e -> e -> t)
         (n m : nat)
         (x : e)
         (advs : nat -> ADV)
         (y : e),
    p x (E (apply_advs (n + m) (leave (n + m)) advs y)) ->
    p x (E (apply_advs n (leave n) advs y)).
Proof.
  intros p n m x advs y H.
  induction m as [| m' IH].
  - rewrite Nat.add_0_r in H. exact H.
  - apply IH.
    assert (H_eq : n + S m' = (n + m') + 1) by lia.
    rewrite H_eq in H.
    apply (MP_leave_layered p (n + m') x advs y). exact H.
Qed.

(* general "adverb-stripping" postulate for verb v *)
Axiom MP_general :
  forall (p : e -> e -> t)
         (v : forall n, IV_ADV n)
         (n : nat)
         (x : e)
         (advs : nat -> ADV)
         (y : e),
    p x (E (apply_advs (n + 1) (v (n + 1)) advs y)) ->
    p x (E (apply_advs n (v n) advs y)).

(* Multi-layered adverb stripping for verb v, remove m layers *)
```

```
Theorem MP_general_multi :
  forall (p : e -> e -> t)
         (v : forall n, IV_ADV n)
         (n m : nat)
         (x : e)
         (advs : nat -> ADV)
         (y : e),
    p x (E (apply_advs (n + m) (v (n + m)) advs y)) ->
    p x (E (apply_advs n (v n) advs y)).
Proof.
  intros p v n m x advs y H.
  induction m as [| m' IH].
  - rewrite Nat.add_0_r in H. exact H.
  - apply IH.
    assert (H_eq : n + S m' = (n + m') + 1) by lia.
    rewrite H_eq in H.
    apply (MP_general p v (n + m') x advs y). exact H.
Qed.
```