Handling Verb Phrase Anaphora with Dependent Types and Events

Daniyar Itegulov[1,2(✉)] and Ekaterina Lebedeva[1]

[1] Australian National University, Canberra, Australia
{daniyar.itegulov,ekaterina.lebedeva}@anu.edu.au
[2] ITMO University, St. Petersburg, Russia

Abstract. This paper studies how dependent typed events can be used to treat verb phrase anaphora. We introduce a framework that extends Dependent Type Semantics (DTS) with a new atomic type for neo-Davidsonian events and an extended @-operator that can return new events that share properties of events referenced by verb phrase anaphora.

The proposed framework, along with illustrative examples of its use, are presented after a brief overview of the necessary background and of the major challenges posed by verb phrase anaphora.

1 Introduction

Davidson [3] observed that some verbs can imply the existence of an "action". For example, the sentence "John eats." represents an action of eating. This action can be anaphorically referred from a following sentence (e.g.: "The food is yummy."). Therefore, it is desirable for framework of natural language semantics to encompass the notion of action and a mechanism for action reference. Davidson proposed to equip interpretations of verbs with an additional argument for events. Thus, the sentence "John eats." is interpreted according to Davidson as $\exists e.eats(e, \mathbf{j})$, instead of $eats(\mathbf{j})$.

Parsons [9] and Taylor [14] argued that the approach of event semantics captures the notion of adverbs better than approaches based on higher-order predicates, such as [15], and is easier to work with. For example, adverbial modifiers usually affect only the event and not the entity, as the following example illustrates:

(1)
  a. John buttered the toast slowly, deliberately, in the bathroom, with a knife, at midnight.
  b. $\exists e.butter(e, \mathbf{j}, \mathbf{t}) \wedge slowly(e) \wedge deliberately(e) \wedge in(e, \mathbf{b}) \wedge \exists k.with(e, k) \wedge at(e, \mathbf{m})$

D. Itegulov—This work is supported by ANU Ph.D. (International) Scholarship and HDR Fee Remission Merit Scholarship provided by the Australian National University.

E. Lebedeva—We thank Bruno Woltzenlogel Paleo and Florrie Verity for numerous helpful discussions and the anonymous reviewers for the constructive feedback.

L. S. Moss et al. (Eds.): WoLLIC 2018, LNCS 10944, pp. 210–222, 2018.
https://doi.org/10.1007/978-3-662-57669-4_12

Sentence (1a) contains adverbs that modify the event of buttering the toast. The corresponding interpretation in events semantics is shown in (1b).

Additionally to adverbial modifiers, Parsons [9] described two more reasons for introducing events as a new atomic type: perception verbs and reference to events.

Parsons [9], furthermore, proposed a framework based on Davidson's event theory, called neo-Davidsonian event semantics, that extends it as follows:

- event participants are introduced via thematic roles
- state verbs, in addition to action verbs, are handled with an abstract variable
- two concepts, event holding and event culmination, are added
- events are decomposed into subevents.

The differences between Davidsonian and neo-Davidsonian approaches can be seen by comparing interpretations (1b) and (2) of Sentence (1a).

(2) $\exists e.butter(e) \wedge agent(e, \mathbf{j}) \wedge patient(e, \mathbf{t}) \wedge slowly(e) \wedge deliberately(e) \wedge in(e, \mathbf{b}) \wedge \exists k.with(e, k) \wedge at(e, \mathbf{m})$

This paper proposes a framework for solving verb phrase anaphora (also known as verb phrase ellipsis) based on the neo-Davidsonian event semantics and on dependent types; and to adapt the existing techniques of handling the propositional anaphora to Dependent Type Semantics (DTS) framework. Dependent types are already used to express pronominal anaphora in [1].

In Sect. 2, we briefly recall Dependent Type Semantics, which is a theoretical foundation for our framework. In Sect. 3, we discuss major challenges of interpreting verb phrase anaphora. The main contribution of this paper is presented in Sect. 4, which describes an extension of the Dependent Type Semantics, and in Sect. 5, which discusses an application of subtyping in the proposed framework.

2 Recalling Dependent Type Semantics

Dynamic type semantics (DTS) proposed by Bekki in [1] is a framework of discourse semantics based on dependent type theory (Martin-Löf [7]). DTS follows the constructive proof-theoretic approach to semantics established by Sundholm [13], who introduced Sundholmian semantics, and by Ranta [12], who introduced Type Theoretical Grammar.

Definition 1 (Dependent function). *For any* $(s_1, s_2) \in \{(type, type), (type, kind), (kind, type), (kind, kind)\}, s \in \{type, kind\}$*:*

$$\dfrac{A : s_1 \qquad \begin{array}{c} x : A \\ \vdots \\ B : s_2 \end{array}}{(x : A) \to B : s_2} \qquad \dfrac{A : s \qquad \begin{array}{c} x : A \\ \vdots \\ M : B \end{array}}{\lambda x.M : (x : A) \to B} \qquad \dfrac{M : (x : A) \to B \qquad N : A}{MN : B[N/x]}$$

Definition 2 (Dependent pair). *For any* $(s_1, s_2) \in \{(type, type), (type, kind), (kind, kind)\}$*:*

$$\frac{A : s_1 \qquad \begin{matrix} x : A \\ \vdots \\ B : s_2 \end{matrix}}{\begin{bmatrix} x : A \\ B \end{bmatrix} : s_2} \quad \frac{M : A \qquad N : B[M/x]}{(M, N) : \begin{bmatrix} x : A \\ B \end{bmatrix}} \quad \frac{M : \begin{bmatrix} x : A \\ B \end{bmatrix}}{\pi_1 M : A} \quad \frac{M : \begin{bmatrix} x : A \\ B \end{bmatrix}}{\pi_2 M : B[\pi_1 M/x]}$$

DTS employs two kinds of dependent types (in addition to simply-typed lambda calculus): *dependent pair type* or Σ-type (notation $(x : A) \rightarrow B(x)$) and *dependent function type* or Π-type (notation $(x : A) \times B(x)$). A dependent pair is a generalization of an ordinary pair. By Curry-Howard correspondence between types and propositions, the type $(x : A) \times B(x)$ corresponds to an existentially quantified formula $\exists x^A.B$ and to an ordinary conjunction $A \wedge B$ when $x \notin fv(B)$.[1] A dependent function is a generalization of an ordinary function; the type $(x : A) \rightarrow B(x)$ corresponds to $\forall x^A.B$ and to $A \rightarrow B$ when $x \notin fv(B)$. Formal definitions are given through inference rules in Definitions 1 and 2.

A comparison between the traditional notation for dependent types and the notation used in DTS can be seen in Fig. 1.

	Π-type	Σ-type
Initial notation	$(\Pi x : A)B(x)$	$(\Sigma x : A)B(x)$
DTS notation	$(x : A) \rightarrow B(x)$	$(x : A) \times B(x), \begin{bmatrix} x : A \\ B(x) \end{bmatrix}$
Initial notation, $x \notin fv(B)$	$A \rightarrow B$	$A \wedge B$
DTS notation, $x \notin fv(B)$	$A \rightarrow B$	$\begin{bmatrix} A \\ B \end{bmatrix}$

Fig. 1. Notation in DTS

The main atomic type in DTS is **entity**, which represents all entities in a discourse. With the employment of dependent type constructors, the entity type can be combined with additional properties. For example, $(\Sigma u : (\Sigma x : \textbf{entity}) \times man(x)) \times enter(\pi_1(u))$ is a valid DTS interpretation of "A man entered". Therefore, in contrast to traditional approaches to semantical interpretation where entities are not distinguished by their types, each entity has its own type in DTS.

In the traditional Montague model-theoretic semantics [8], a proposition denotes a truth value (often defined as an o-type). However, DTS does not follow this convention and instead the meaning of a sentence is represented by

[1] $fv(x)$ denotes all free variables in x.

a type. Types in DTS are defined by the inference rules, as shown in Definitions 1 and 2. The rules specify how a dependent type (as a proposition) can be proved under a given context. Thus, the meaning of a sentence in proof-theoretic semantics lies in its *verification condition* similar to the philosophy of language by Dummett [4,5] and Prawitz [10].

To handle anaphora resolution, DTS distinguishes two kinds of propositions: static and dynamic. A static proposition P is called true if it is inhabited, i.e. there exists a term of type P. A dynamic proposition is a function mapping context-proof (a static proposition that is an interpretation of the previous discourse) to a static proposition.

In order to represent anaphoric references and presupposition triggers, Bekki introduced a @-operator. The operator takes the left context of dynamic propositions in which it occurs. For example, Sentence (3a) can be interpreted as (3b) in DTS. The @-operators in (3b) take a context as an argument and try to find a female (due to the interpretation of the word "herself") entity in the context passed to them.

@-operators have a single introduction rule $@F$ (Fig. 2):

$$\frac{A : \texttt{type} \qquad A\ \mathit{true}}{(@_i : A) : A}$$

Fig. 2. $@F$ introduction rule

Different @-operators can take contexts of different types. Therefore, they can be of different types and are distinguished with a numerical subscript. The full type of the $@_i$-operator can look like this: $@_i : \gamma_i \to \mathbf{entity}$.

(3)
a. She loves herself.
b. $\lambda c.loves(\pi_1(@_1 c : \begin{bmatrix} x : \mathbf{entity} \\ female(x) \end{bmatrix}), \pi_1(@_2 c : \begin{bmatrix} x : \mathbf{entity} \\ female(x) \end{bmatrix}))$

The *felicity condition* states that in order to be felicitous, an instance of a syntactic category S (i.e. a sentence) has to be of sort $\gamma \to \mathbf{type}$ for some new variable γ. To check the felicity condition of a sentence, a type checking algorithm must be evoked. Consider that the interpretation of the sentence "Mary lives in London" is passed to (3b) as the left context. The result is shown in (4).

(4) $$\begin{bmatrix} v : \begin{bmatrix} u : \begin{bmatrix} \mathbf{m} : \mathbf{entity} \\ female(\mathbf{m}) \end{bmatrix} \\ lives\_in(\pi_1(u), \mathbf{l}) \end{bmatrix} \\ \lambda c.loves(\pi_1(@_1 c : \begin{bmatrix} x : \mathbf{entity} \\ female(x) \end{bmatrix}), \pi_1(@_2 c : \begin{bmatrix} x : \mathbf{entity} \\ female(x) \end{bmatrix})) \end{bmatrix}$$

However, before type-checking this sentence, @-operators should be built using the $@F$ introduction rule. The rule requires the existence of a proof term inhabiting the type of the @-operator. In case of Sentence (4), both $@_1$ and $@_2$

operators are inhabited by the proof term $\lambda c.\pi_1(c)$. In particular, Mary (the first element of the dependent pair in the context) is a valid entity having the property of being female for both pronouns "she" and "herself". Hence, the anaphora resolution process involves proof search and can be done using theorem provers.

3 Verb Phrase Anaphora

Verb phrase anaphora [11] are anaphora with an intentional omission of part of a full-fledged verb phrase when the ellipsed part can be implicitly derived from the context. For example, verb phrase anaphora can be observed in (5a) and (5b):

(5)
a. John left before Mary did.
b. John left. Mary did too.

In (5a), the word "did" refers to an action John did before Mary. In (5b), the "did too" clause refers to an action which John and Mary both did. These sentences can be interpreted in event semantics as the following logical expressions:

(6)
a. $\exists e.agent(e, \mathbf{j}) \wedge left(e) \wedge \exists e'.agent(e', \mathbf{m}) \wedge left(e') \wedge before(e, e')$
b. $\exists e.agent(e, \mathbf{j}) \wedge left(e) \wedge \neg\exists e'.agent(e', \mathbf{m}) \wedge left(e')$

Furthermore, an anaphoric verb phrase can "inherit" some properties from its referent. Consider Example (7a) where "did too" not only refers to the event of eating performed by John, but also to properties such as "quietly" and "last night". Expression (7b) is the interpretation of this sentence in event semantics.

(7)
a. John quietly ate the cake last night. Mary did too.
b. $\exists e.(agent(e, \mathbf{j}) \wedge patient(e, \mathbf{c}) \wedge ate(e) \wedge quietly(e) \wedge at(e, \mathbf{ln})) \wedge$
$\exists e'.(agent(e', \mathbf{m}) \wedge patient(e', \mathbf{c}) \wedge ate(e') \wedge quietly(e') \wedge at(e', \mathbf{ln}))$

A verb phrase anaphor may have an additional property that can ease the choice of a correct anaphoric referent-event from the context. This phenomenon is exemplified in (8a), where it is explicit that "too" refers to an action connected with eating.

(8) a. John ate pasta and did not feel well. Mary ate too, but nothing happened to her.

An ambiguity between strict and sloppy identity readings of verb phrase anaphora described by Prüst [11] is another intriguing phenomenon. Example (9) illustrates this:

(9) a. John likes his hat. Fred does too.
b. $\exists x.hat(x) \wedge owner(x, \mathbf{j}) \wedge \exists e.like(e) \wedge agent(e, \mathbf{j}) \wedge patient(e, x) \wedge$
$\exists e'.like(e') \wedge agent(e', \mathbf{f}) \wedge patient(e', x)$

c. $\exists x.hat(x) \wedge owner(x, \mathbf{j}) \wedge \exists e.like(e) \wedge agent(e, \mathbf{j}) \wedge patient(e, x) \wedge \exists y.hat(y) \wedge owner(y, \mathbf{f}) \wedge \exists e'.like(e') \wedge agent(e', \mathbf{f}) \wedge patient(e', y)$

The anaphoric clause in the second sentence of (9a) can be interpreted as "Fred likes John's hat" (the sloppy identity interpretation (9b)) or as "Fred likes Fred's hat" (the strict interpretation (9c)). A desirable framework should be able to provide both interpretations.

4 Events with Dependent Types

To tackle phenomena discussed in Sect. 3, we propose to extend DTS with a new atomic type **event** for interpreting events. Then, given its left context c, DTS's @-operator can be employed for retrieving a variable of type **event** analogously to its original use for retrieving a referent of type **entity**.

As was shown by Parsons [9], event semantics can be employed to represent propositional anaphora. An example of propositional anaphora is shown in (10a), where "this" refers to the whole proposition expressed in the first sentence. Formula (10b) is an interpretation of (10a).

(10)
a. John loved Mary. But Mary did not believe this.
b. $\exists e.agent(e, \mathbf{j}) \wedge patient(e, \mathbf{m}) \wedge loved(e) \wedge \exists e'.believed(e') \wedge agent(e', \mathbf{m}) \wedge patient(e', e)$

Dependent typed events allow us to handle more complex types of propositional anaphora. Similar to entities, events can have various properties provided by their description. Assume the following three sentences appear in the same discourse, possibly remotely from each other, but with preservation of the order:

(11)
a. Canberra was hit by a flood on Sunday.
b. The fair was held in London.
c. What happened in Canberra is surprising.

Here the anaphoric clause in Sentence (11c) refers to an event discussed earlier. There are however (at least) two potential events for the reference: one given by (11a) and another given by (11b). Since the anaphoric clause in (11c) specifies that it refers to an event happened in Canberra, the anaphor disambiguates to the event in (11a).

The interpretation of verb phrase anaphora is more challenging, however, than the interpretation of propositional anaphora: an anaphoric clause in a verb phrase usually talks about a new event that inherits properties of another event. For example, "John left. Bob did too." conveys two events: one is about John leaving and the second one is about Bob leaving. In cases of pronominal and propositional anaphora, however, there is just a reference to an entity or an event in the context. For example in "John walks. He is slow.", pronoun "he" in the second sentence just refers to the entity "John" from the first sentence.

To handle verb phrase anaphora correctly, it is not enough to just fetch a referenced variable from the left context; instead a new variable of type **event** should be introduced. This new variable *copies* properties from the referred event. Furthermore, the agent of the referred event should be changed to the current agent in the new event. This can be seen in interpretation (7b) of (7a), where the agent John is replaced with Mary.

Although $@_i$-operator has type $\gamma_i \rightarrow \textbf{entity}$ in DTS for handling pronominal anaphora, according to DTS syntax for raw terms the operator can be of any type. We therefore suggest a new type of @-operator that guarantees that the returned event has a proper agent, necessary for interpreting verb phrase anaphora:

(12) $$@_i : (c : \gamma_i) \rightarrow (x : \textbf{entity}) \rightarrow \begin{bmatrix} e : \textbf{event} \\ agent(e, x) \end{bmatrix}$$

Formula (13a) is an interpretation of discourse (5b). The $@_1$-operator in (13b) is applied to its left context c (of type γ_0) and an entity, and returns a *new* event of type **event** with the same properties (apart from the agent property) of the referenced event. Crucially, the event returned by $@_1$-operator in (13b) is not an event that was in the context previously. It is a new event with the same properties (e.g. location, time) as a referenced event from the context, but with a replaced agent.

(13)

a. $$\lambda c. \begin{bmatrix} e : \textbf{event} \\ \begin{bmatrix} left(e) \\ agent(e, \mathbf{j}) \end{bmatrix}) \end{bmatrix}$$

b. $$\lambda c.(@_1 c : (x : \textbf{entity}) \rightarrow \begin{bmatrix} e : \textbf{event} \\ agent(e, x) \end{bmatrix})(\mathbf{m})$$

Note that the entity accepted by the @-operator defined in (12) is the agent in the new event. For instance, in Example (7a), the interpretation of "did too" using (12) would have the agent "John" of the referenced event replaced by "Mary", but the patient (i.e. the cake) would remain. On the other hand, there exist cases of verb phrase anaphora where the patient in the referenced event should be replaced. This usually depends on the voice (active or passive) of an anaphoric clause, as can be seen from examples in (14).

(14)

a. Mary is loved by John. So is Ann.

b. John loves Mary. So does Bob.

c. $$\begin{bmatrix} u : \begin{bmatrix} e : \textbf{event} \\ \begin{bmatrix} agent(e, \mathbf{j}) \\ \begin{bmatrix} patient(e, \mathbf{m}) \\ loved(e) \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} e' : \textbf{event} \\ \begin{bmatrix} agent(e', \mathbf{j}) \\ \begin{bmatrix} patient(e', \mathbf{a}) \\ loved(e') \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

d. $$\begin{bmatrix} u : \begin{bmatrix} e : \textbf{event} \\ \begin{bmatrix} agent(e, \mathbf{j}) \\ \begin{bmatrix} patient(e, \mathbf{m}) \\ loved(e) \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} e' : \textbf{event} \\ \begin{bmatrix} agent(e', \mathbf{b}) \\ \begin{bmatrix} patient(e'''', \mathbf{m}) \\ loved(e') \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The first sentences in (14a) and (14b) have the same semantics and hence the interpretations given to them in (14c) and (14d) coincide. However, despite the fact that both second sentences are written in the same voice as their first sentences, the second sentences are interpreted differently. Naturally, the second sentence in (14a) means "Ann is loved by John", while the second sentence in (14b) means "Bob loves Mary". Note that they have replaced different participants of the first sentences: in (14a) Mary (patient) was replaced by Ann and in (14b) John (agent) was replaced by Bob.

Furthermore, the interpretation of a sentence may require both the agent and the patient to be replaced, as for example in the sloppy reading of (9c). These possible cases of anaphora resolution can be tackled with the judgements (16) assuming they occur in a global context $\mathcal{K}$.

Another important notion in DTS is the felicity condition. The anaphora resolution for $@_i$ operator is launched by type checking of the following judgement: $\mathcal{K}, \gamma_i : type \vdash @_i : \gamma_i \rightarrow type$. It means that the semantical interpretation of a sentence must be of the sort `type` assuming that the left context is of type γ_i. A requirement of a success of the launching the type checker is called *felicity condition.*

In order to preserve the original DTS invariants, we should show how the felicity condition is being fulfilled in the extended DTS. An example of a felicity-judgement generated by verb phrase anaphora is shown in example (15). It is different from the felicity condition from original DTS notion since the new @-operator has a new type as shown in (12).

(15) $\mathcal{K}, \gamma_i : type \vdash @_i : \gamma_i \rightarrow (x : \mathbf{entity}) \rightarrow \begin{bmatrix} e : \mathbf{event} \\ agent(e, x) \end{bmatrix}$

Assume that the global context $\mathcal{K}$ contains the judgements from (16). Then one should be able to type check judgements generated by verb phrase anaphora.

(16)

a. $replaceA : (p : \mathbf{entity} \rightarrow (e : \mathbf{event}) \rightarrow \mathbf{type}) \rightarrow (original : \mathbf{entity}) \rightarrow (new : \mathbf{entity}) \rightarrow (u : \begin{bmatrix} e' : \mathbf{event} \\ p\ original\ e' \end{bmatrix}) \rightarrow (v : \begin{bmatrix} e'' : \mathbf{event} \\ p\ new\ e'' \end{bmatrix})$

b. $replaceP : (p : \mathbf{entity} \rightarrow (e : \mathbf{event}) \rightarrow \mathbf{type}) \rightarrow (original : \mathbf{entity}) \rightarrow (new : \mathbf{entity}) \rightarrow (u : \begin{bmatrix} e' : \mathbf{event} \\ p\ original\ e' \end{bmatrix}) \rightarrow (v : \begin{bmatrix} e'' : \mathbf{event} \\ p\ new\ e'' \end{bmatrix})$

c. $replaceAP : (p : \mathbf{entity} \rightarrow \mathbf{entity} \rightarrow (e : \mathbf{event}) \rightarrow \mathbf{type}) \rightarrow (oagent : \mathbf{entity}) \rightarrow (nagent : \mathbf{entity}) \rightarrow (opatient : \mathbf{entity}) \rightarrow (npatient : \mathbf{entity}) \rightarrow (u : \begin{bmatrix} e' : \mathbf{event} \\ p\ oagent\ opatient\ e' \end{bmatrix}) \rightarrow (v : \begin{bmatrix} e'' : \mathbf{event} \\ p\ nagent\ npatient\ e'' \end{bmatrix})$

d. $j : \mathbf{entity}$

Functions *replaceA*, *replaceP*, *replaceAP* construct a new event v from an existing event u. To express the inheritance of properties and the change of the agent in *replaceA* (or patient in *replaceP*), properties are expressed as a function that accepts two arguments: an agent-entity (or patient-entity in *replaceP*) and an event; and returns a logical expression describing the event using the entity. Function *replaceAP* accounts for cases where both an agent and a patient are replaced.

We can now construct term $@_1$ of type (12), to fulfill the felicity condition of form (15), as shown in (17):

$$(17)\ \mathcal{K}, \gamma_0 : type \vdash @_1 : \gamma_0 \to (x : \mathbf{entity}) \to \begin{bmatrix} e : \mathbf{event} \\ agent(e, x) \end{bmatrix} = \lambda c.\lambda x.replaceA\ (\lambda y.\lambda e. \begin{bmatrix} left(e) \\ agent(e, y) \end{bmatrix})\ \mathbf{j}\ x\ \pi_1\pi_2(c)$$

A substitution of $@_1$ in (13b) with its term defined in (17) leads to the following semantical interpretation:

$$(18)\ \begin{bmatrix} e'' : \mathbf{event} \\ \begin{bmatrix} left(e'') \\ agent(e'', \mathbf{m}) \end{bmatrix} \end{bmatrix}$$

Since anaphora in DTS are resolved using the type checking procedure, verb phrase anaphora, just like pronominal anaphora, can be resolved in various ways. A type checking algorithm can find different terms which conform to the specified (by felicity condition) type. For example, in order to handle the ambiguity between strict and sloppy identity readings, which were discussed in Example (9), our framework can provide both possible interpretations for Sentence (9a). Term (19) shows a generic interpretation of (9a) in the proposed framework.

$$(19)\ \lambda c. \begin{bmatrix} u : \begin{bmatrix} v : \begin{bmatrix} x : \mathbf{entity} \\ \begin{bmatrix} hat(x) \\ owner(x, \mathbf{j}) \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} e : \mathbf{event} \\ \begin{bmatrix} like(e) \\ \begin{bmatrix} agent(e, \mathbf{j}) \\ patient(e, \pi_1(v)) \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ @_1(c, u)\mathbf{f} : \begin{bmatrix} e' : \mathbf{event} \\ agent(e', \mathbf{f}) \end{bmatrix} \end{bmatrix}$$

In (19), "Fred does too." is interpreted as the term $@_1(c, u)\mathbf{f}$, where u stands for the interpretation of the preceding sentence "John likes his hat.". Recall from Example (9) that the latter sentence has an ambiguous meaning. (20) defines two alternative terms for $@_0$, one for each of the possible meanings. Note that the type of these terms for $@_0$ conforms with the felicity condition.

(20)

a. $\mathcal{K} \vdash @_1 : \gamma_0 \to (x : \textbf{entity}) \to \begin{bmatrix} e' : \textbf{event} \\ agent(e', x) \end{bmatrix} =$

$$\lambda c.\lambda f.replaceA\ (\lambda y.\lambda e. \begin{bmatrix} like(e) \\ \begin{bmatrix} agent(e, y) \\ patient(e, x) \end{bmatrix} \end{bmatrix})\ \mathbf{j}\ f\ \pi_1\pi_2\pi_2(c)$$

b. $\mathcal{K} \vdash @_1 : \gamma_0 \to (x : \textbf{entity}) \to \begin{bmatrix} e' : \textbf{event} \\ agent(e', x) \end{bmatrix} =$

$$\textbf{let}\ p = \lambda y.\lambda z.\lambda e. \begin{bmatrix} like(e) \\ \begin{bmatrix} agent(e, y) \\ patient(e, z) \end{bmatrix} \end{bmatrix}$$

$$\textbf{in}\ \lambda c.\lambda f. \begin{bmatrix} u : \begin{bmatrix} y : \textbf{entity} \\ hat(y) \wedge owner(y, f) \end{bmatrix} \\ replaceAP\ p\ \mathbf{j}\ f\ \pi_1\pi_1\pi_2(c)\ \pi_1(u)\ \pi_1\pi_2\pi_2(c) \end{bmatrix}$$

Both terms are valid substitutions for $@_1$-operator in (19) and they represent strict and sloppy anaphora readings respectively. In (20b) let-in structure is used only as a syntactical sugar for readability and is not actually a part of DTS term syntax.

In line with the original approach of Bekki [1], the verb phrase anaphora resolution for @-operator involves proof search and can be done using a theorem prover.

5 Subtyping

The equation in (17) (i.e. an anaphora resolution solution: a proof of existence of a term with the required type under the global context $\mathcal{K}$) is not sound: the type of the right side of the equation is

$$\gamma_0 \to (x : \textbf{entity}) \to \begin{bmatrix} e'' : \textbf{entity} \\ \begin{bmatrix} left(e'') \\ agent(e'', x) \end{bmatrix} \end{bmatrix}$$

while the type required by the left side is

$$\gamma_0 \to (x : \textbf{entity}) \to \begin{bmatrix} e : \textbf{entity} \\ agent(e, x) \end{bmatrix}$$

The former type is more specific than the latter type because it has the additional property "left".

This is not a problem, as events have a natural subtyping relationship between them. As described by Luo and Soloviev in [6], an event whose agent is a and patient is p, is an event with agent a. Despite a different theory underneath, the techniques described there can be reused for subtyping events in DTS. This leads to the following subtyping relations in event semantics:

(21)
a. $Evt_{AP}(a,p) <: Evt_A(a) <: Event \longleftrightarrow$

$$\begin{bmatrix} e : \textbf{event} \\ \begin{bmatrix} agent(e,a) \\ patient(e,p) \end{bmatrix} \end{bmatrix} <: \begin{bmatrix} e : \textbf{event} \\ agent(e,a) \end{bmatrix} <: \begin{bmatrix} e : \textbf{event} \\ () \end{bmatrix}$$

b. $Evt_{AP}(a,p) <: Evt_P(p) <: Event \longleftrightarrow$

$$\begin{bmatrix} e : \textbf{event} \\ \begin{bmatrix} agent(e,a) \\ patient(e,p) \end{bmatrix} \end{bmatrix} <: \begin{bmatrix} e : \textbf{event} \\ patient(e,p) \end{bmatrix} <: \begin{bmatrix} e : \textbf{event} \\ () \end{bmatrix}$$

Subtyping relations of events can also depend on other properties (e.g. a loud event performed by John is also an event performed by John). We employ Luo and Soloviev's notation to define a new type $Event_{NA}(n,a)$, which is the type of events with agent a and nature n. Nature is a main predicate for each event in neo-Davidsonian semantics (e.g. "left(e)" for an event of leaving, "ate(e)" for an event of eating).

The following transformation shows how the dependent event types in DTS notation from (17) can be converted into dependent event types in the notation of Luo and Soloviev:

(22)
a. $$\begin{bmatrix} e'' : \textbf{event} \\ \begin{bmatrix} left(e'') \\ agent(e'',x) \end{bmatrix} \end{bmatrix} \longleftrightarrow e'' : Event_{DA}(left,x)$$

b. $$\begin{bmatrix} e : \textbf{entity} \\ agent(e,x) \end{bmatrix} \longleftrightarrow e : Event_A(x)$$

A subtyping relationship between these types can be constructed (assuming the appropriate subtyping rules have been added along with type $Event_{DA}(d,a)$).

$$\frac{left : Description \qquad x : Agent}{Event_{DA}(left,x) <: Event_A(x)}$$

The discussed subtyping relationship allows us to obtain (17).

6 Comparison with Previous Approaches

The approach presented here shares the goal of interpreting elliptical constructions, verb phrase ellipses in particular, with the work of Dalrymple et al. [2]. However, there are crucial conceptual differences.

The method of Dalrymple, Shieber and Pereira relies on the parallel structure of sentences involved in verb phrase anaphora (e.g.: "John loves golf" and "Bob does too" are parallel in the sense that the second sentence can be used as a verb phrase anaphoric clause referring to the first sentence). They rely on a black-box mechanism to determine the parallel structures.

Here we rely on a theorem prover to find a proof term for @-operators. In this way we leverage the advances in the extensively researched field of automated theorem proving.

Our approach, furthermore, avoids some problems described in [2]. For example, it supports semantic parallelism, i.e. parallelism between a "logical subject" and an anaphoric clause. Consider Sentence (23), which is a slightly simplified version of sentence (59) from [2].

(23) The material can be presented in an accessible fashion, and often I do.

We interpret the subject in the first part of the sentence as an existentially quantified variable of type **entity**. This variable can be accessed from the second sentence. In other words, our approach introduces a logical subject that can be used for interpreting an anaphoric clause that follows.

7 Conclusion

This paper introduces dependent event types for resolving verb phrase anaphora with DTS as the underlying framework. To tackle verb phrase anaphora, we extend DTS's @-operator, which was originally introduced for handling pronominal anaphora. The paper also addresses strict and sloppy readings of verb phrase anaphora and shows that each of them can be achieved solely by manipulating the interpretation of the @-operator. The previous approaches to handling the propositional anaphora were also adapted to DTS framework.

Techniques described in this paper could be applied to handle other cases of anaphora, such as adjectival anaphora, modal and "do so" anaphora. Another interesting topic would be to study specific behaviours of various thematic roles, such as experiencer, theme and source.

References

1. Bekki, D.: Representing anaphora with dependent types. In: Asher, N., Soloviev, S. (eds.) LACL 2014. LNCS, vol. 8535, pp. 14–29. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43742-1_2
2. Dalrymple, M., Shieber, S.M., Pereira, F.C.N.: Ellipsis and higher-order unification. Linguist. Philos. **14**(4), 399–452 (1991)
3. Davidson, D.: The logical form of action sentences. In: Rescher, N. (ed.) The Logic of Decision and Action. University of Pittsburgh Press, Pittsburgh (1967)
4. Dummett, M.: What is a theory of meaning? (II). In: Evans, G., McDowell, J. (eds.) Truth and Meaning: Essays in Semantics. Clarendon Press, Oxford (1976)
5. Dummett, M.A.E.: What is a theory of meaning? In: Guttenplan, S. (ed.) Mind and Language. Oxford University Press, Oxford (1975)
6. Luo, Z., Soloviev, S.: Dependent event types. In: Kennedy, J., de Queiroz, R.J.G.B. (eds.) WoLLIC 2017. LNCS, vol. 10388, pp. 216–228. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-55386-2_15
7. Martin-Löf, P., Sambin, G.: Intuitionistic Type Theory. Studies in Proof Theory. Bibliopolis, Berkeley (1984)

8. Montague, R.: Formal Philosophy; Selected Papers of Richard Montague. Yale University Press, New Haven (1974)
9. Parsons, T.: Events in the Semantics of English: A Study in Subatomic Semantics. MIT Press, Cambridge (1990)
10. Prawitz, D.: Intuitionistic logic: a philosophical challenge. In: Von Wright, G.H. (ed.) Logic and Philosophy/Logique et Philosophie, vol. 5, pp. 1–10. Springer, Dordrecht (1980). https://doi.org/10.1007/978-94-009-8820-0_1
11. Prüst, H., Scha, R., van den Berg, M.: Discourse grammar and verb phrase anaphora. Linguist. Philos. **17**(3), 261–327 (1994)
12. Ranta, A.: Type-Theoretical Grammar. Oxford University Press, Oxford (1994)
13. Sundholm, G.: Proof theory and meaning. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic: Volume III: Alternatives in Classical Logic. SYLI, vol. 166, pp. 471–506. Springer, Dordrecht (1986). https://doi.org/10.1007/978-94-009-5203-4_8
14. Taylor, B.: Modes of occurence, verbs, adverbs and events. Revue Philosophique de la France Et de l'Etranger **176**(3), 406–407 (1986)
15. Verkuyl, H.J.: On the Compositional Nature of the Aspects. D. Reidel Publishing Company, Dordrecht (1972)