NELS 45: Proceedings of the 45th Meeting of the North East Linguistic Society
(ed. Thuy Bui & Deniz Özyıldız). Volume 3: 261— 270.

Decomposing English particles *and* and *or*[*]

Linmin Zhang

New York University

1. Introduction

As (1) and (2) show, the classical view on the semantics of *and* and *or* is that they are natural language counterparts of boolean operators $\sqcap$ and $\sqcup$ (Montague (1973), von Stechow (1974), Gazdar (1980), Partee & Rooth (1983), Champollion (2015), etc.).

(1) *and* is analyzed as an intersective operator, defined in terms of conjunction:

a. $[\![\text{and}_{\sqcap}]\!] = \sqcap_{\langle\tau,\tau\tau\rangle} =_{def} \begin{cases} \wedge_{\langle t,tt\rangle} & \text{if } \tau = t \\ \lambda X_\tau \lambda Y_\tau \lambda Z_{\sigma_1}.X(Z) \sqcap_{\langle\sigma_2,\sigma_2\sigma_2\rangle} Y(Z) & \text{if } \tau = \langle\sigma_1,\sigma_2\rangle \end{cases}$

b. Al and Jo came to the party. $[\![\text{Al and Jo}]\!] = \lambda P.\text{Al}(P) \sqcap \text{Jo}(P)$

(2) *or* is analyzed as a union operator, defined in terms of disjunction:

a. $[\![\text{or}_{\sqcup}]\!] = \sqcup_{\langle\tau,\tau\tau\rangle} =_{def} \begin{cases} \vee_{\langle t,tt\rangle} & \text{if } \tau = t \\ \lambda X_\tau \lambda Y_\tau \lambda Z_{\sigma_1}.X(Z) \sqcup_{\langle\sigma_2,\sigma_2\sigma_2\rangle} Y(Z) & \text{if } \tau = \langle\sigma_1,\sigma_2\rangle \end{cases}$

b. Al or Jo came to the party. $[\![\text{Al or Jo}]\!] = \lambda P.\text{Al}(P) \sqcup \text{Jo}(P)$

Alternatively, as (3) shows, *and* is sometimes analyzed as a sum operator $\oplus$ (Krifka (1990), Munn (1993), Lasersohn (1995), Heycock & Zamparelli (2005), etc.).

(3) *and* is analyzed as a sum operator, which sums up entities into a plural entity:

a. $[\![\text{and}_{\oplus}]\!] = \oplus_{\langle e,ee\rangle} =_{def} \lambda x_e \lambda y_e.x \oplus y$

b. Al and Jo met here. $[\![\text{Al and Jo}]\!] = \text{Al} \oplus \text{Jo}$

[*]I thank Lucas Champollion and Yoad Winter for suggestions and discussions through various stages of this project. I also thank Maria Aloni, Chris Barker, Dylan Bumford, Simon Charlow, Ivano Ciardelli, Kit Fine, Sonia Kasyanenko, Manuel Križ, Jeremy Kuhn, David Nicolas, Jim Pryor, Philippe Schlenker, Anna Szabolcsi, Yimei Xiang and the reviewers of NELS45 for comments. Of course, none of these people necessarily agree with the view of this paper.

In this paper, I argue that both the classical boolean view and the sum operator analysis under- and over-generate. Following the insight of Winter (1995, 1998) and Szabolcsi (2015), as well as the analysis of Munn (1993) and Winter (2006), I propose that natural language coordinators *and* and *or* are essentially markers of the data structure **list**.

§2 presents the empirical motivation, focusing on the data involving order-sensitivity (and repetition) and multiple coordination, which challenge both the boolean view and the sum operator analysis. §3 presents the details of the current proposal: the syntactic and semantic contribution of *and* and *or* is composed of three pieces – (i) a list-building operator, (ii) a deterministic (in the case of *and*) or nondeterministic (in the case of *or*) operator which filters the list, and (iii) a fold operator which makes the list ready for further semantic composition. §4 shows how the current proposal accounts for the data discussed in §2. §5 concludes the paper and suggests avenues for further research.

2. Empirical motivation: order-sensitivity and multiple coordination

Here I show that the boolean view and the sum operator analysis (i) under-generate and fail to account for some data of *and* with a repetition of coordinated items and showing the order-sensitivity and (ii) over-generate unattested patterns of multiple coordination (MC).

2.1 Order-sensitivity and repetition in using *and*

The operations of intersection and sum are commutative (i.e., $A \sqcap B = B \sqcap A$; $A \oplus B = B \oplus A$) and idempotent (i.e., $A \sqcap A = A$; $A \oplus A = A$).[1] Thus, the analyses of *and* shown in (1) and (3) predict that *A and B* and *B and A* always have the same meaning, and that such repetition expressions as *A and A* are ruled out due to redundancy.

However, these predictions are not borne out, as illustrated in (4). The truth conditions of (4a) and (4b) are clearly different, and thus (4c) is not a weird sentence. Moreover, as (4d) shows, with the use of (i) an order-sensitive definite plural (e.g., here *the **last three** NP*) as well as (ii) a conjunction with a repetition of items (e.g., here *Nadal, Federer and Nadal*), the sentence can hardly be followed by *but not necessarily in that order*, suggesting that the order-sensitivity shown in the expression *Nadal, Federer and Nadal* is not an implicature. This is even more evident when the word *respectively* is explicitly uttered; when *respectively* is not uttered (as in (4a)-(4c)), the existence of such a silent word is inferred.

(4) a. The last three champions at Roland Garros were $\underline{\text{Nadal, Federer and Nadal}}$.[2]
b. The last three champions at Roland Garros were $\underline{\text{Nadal, Nadal and Federer}}$.
c. The last three champions at Roland Garros were not $\underline{\text{Nadal, Federer and Nadal}}$, but $\underline{\text{Nadal, Nadal and Federer}}$.
d. The last three champions at Roland Garros were $\underline{\text{Nadal, Federer and Nadal}}$ (**respectively**), # but not necessarily in that order.

In sum, $\sqcap$ and $\oplus$ are commutative and idempotent, but *and* is not necessarily so.

[1]See Tarski (1935) for the properties of the sum operation in classical extensional mereology (CEM).
[2]The sentence (4a) is taken from Florio & Nicolas (2014).

2.2 Multiple coordination (MC)

(1)-(3) also show that in both the boolean and the sum operator analyses, *and/or* takes arguments of the same type and returns an output of that type. Here I show that this inevitably over-generates unattested patterns in the case of MC.

To begin with, (5) illustrates how coordinators are actually used in MC: either the coordinator only occurs before the rightmost item (see (5a)), or it occurs between any two adjacent items (see (5b)); other cases are unacceptable (see (5c) and (5d)).

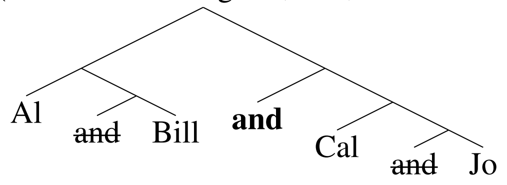
(5) a. ✓ A, B **and** C came. c. * A **and** B, C came.
b. ✓ A **and** B **and** C came. d. * A, B, C came.

Two hypotheses are discussed in Winter (2006) to account for the distribution of coordinators in MC. Hypothesis 1: *and* and *or* (i) are binary operators (as in (1) and (2)) and (ii) can be silent. Under this hypothesis, *and* and *or* are recursively used in MC, and MC is built in a nested way – e.g., [[*A* ~~*and*~~ *B*] *and C*]. A stipulation is needed here: *and/or* can freely go silent except in its last application. Winter (2006) shows that this hypothesis over-generates unattested readings: it predicts that (6a) and (6b) are semantically equivalent and share all the possible readings; however, with a certain intonation (see the bracketing), (6a) can have a '2 requirements' reading, which is by no means available for (6b).

(6) a. You need to [[dance and jump] and sing]. two readings available
✓ 2 requirements (dance & jump; sing); ✓ 3 requirements (dance; jump; sing)
b. You need to dance, jump and sing. only one reading available
# 2 requirements; ✓ 3 requirements (dance; jump; sing)

Another problem of this hypothesis (which is not mentioned in Winter (2006)) is that it also over-generates ill-formed expressions, as illustrated in (7). Crucially, such an over-generation is inevitable if the type of the coordinator is of the pattern $\langle\alpha,\alpha\alpha\rangle$.

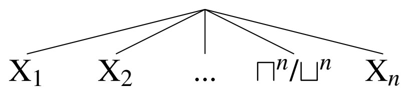
(7) a. *Al, Bill, and Cal, Jo smiled. (intended meaning: Al, Bill, Cal and Jo smiled.)
b. LF of the MC part:



In Winter (2006), Hypothesis 2 proposes that binary operators *and* and *or* can be generalized into *n*-ary operators with a recursive definition (see (8a) and (8b)), so that MC is built with a flat structure, as (8c) shows. Also a stipulation is needed here: *and* and *or* are never silent in MC, but always pronounced right before the rightmost item.

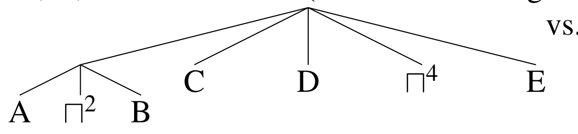
(8) a. $\sqcap^n_{\langle\tau^1\langle\tau^2...\langle\tau^n\tau\rangle\rangle\rangle} =_{def} \lambda X^1_\tau...\lambda X^n_\tau.\sqcap^2(X^1,\sqcap^{n-1}(X^2,...,X^n))$
b. $\sqcup^n_{\langle\tau^1\langle\tau^2...\langle\tau^n\tau\rangle\rangle\rangle} =_{def} \lambda X^1_\tau...\lambda X^n_\tau.\sqcup^2(X^1,\sqcup^{n-1}(X^2,...,X^n))$

c. LF of a MC construction:

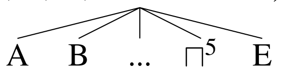


Based on the empirical evidence shown in (6b) (i.e., the MC sentence expresses three semantically equal requirements), Winter (2006) argues that Hypothesis 2 is superior to Hypothesis 1: semantically, MC should be composed in a flat way with the use of *n*-ary operators. However, here I show that Hypothesis 2 also over-generates: such a sentence as (9) could have been generated via the use of a 4-ary operator $\sqcap^4$ plus a binary operator $\sqcap^2$, because there is no constraint forcing the use of a 5-ary operator. Crucially, this over-generation is unavoidable if the types of *and*/*or* are of the pattern $\langle\alpha,\langle\alpha,...\rangle\rangle$.

(9) a. *A, and B, C, D and E smiled. (intended meaning: A, B, C, D, and E smiled.)
b. LF:



vs.



In sum, so long as *and* and *or* are analyzed as operators taking arguments of the same type and returning an output of that type (i.e., of the type $\langle\alpha,\alpha\alpha\rangle$ or $\langle\alpha,\langle\alpha,...\rangle\rangle$), over-generation is inevitable, and the distribution of *and*/*or* in MC remains unaccounted for.

3. Proposal: decomposing English particles *and* and *or* into three operators

The discussion in §2 suggests that natural language words *and* and *or* cannot be analyzed as such commutative and idempotent operators as $\sqcap$, $\sqcup$ and $\oplus$ themselves. This interim conclusion is consistent with the insight originally developed in Winter (1995, 1998) and recently revived in Szabolcsi (2015): based on cross-linguistic data, Szabolcsi (2015) argues that it is common in natural languages that pronounced particles do not perform boolean operations or sum up entities themselves, but point to the existence of silent operators which actually perform boolean operations or sum up entities.

Furthermore, the discussion of MC data in §2.2 is consistent with Munn (1993) and Winter (2006) in showing two seemingly contradictory properties in using natural language words *and* and *or*:[3] on the one hand, the distribution of coordinators in MC suggests that a purely flat and symmetric analysis (i.e., all coordinated items are taken to be structurally equal) cannot accurately characterize the **syntax** in using *and*/*or*; on the other hand, the meaning of (6b) suggests that all coordinated items in MC are **semantically** equal.

Therefore, a sufficiently good analysis of *and* and *or* needs to account for (i) the syntactic asymmetry and (ii) the semantic symmetry in using coordinators, as well as (iii) the order-sensitivity in using *and*. To achieve these goals, a natural move is to follow the recent trend relating natural language semantics with the ways computer scientists (especially functional programmers) reason about computations (see also Charlow (2014), Barker & Shan (2014), and Bumford (2015), etc.), and to adopt the concept **list** (such as the data type list – $[\alpha]$ – used in Haskell) in analyzing the syntax and semantics of coordination.

[3]See Munn (1993) and Winter (2006) for other data and a more extensive discussion on these properties.

A list, e.g., $[5,2,2,...]$, represents a sequence of values, where (i) values are of the same type and (ii) the order among the items in a list is recorded and thus the same value may occur more than once. If the values of a list are of the type α, then the type of the list is $[\alpha]$.

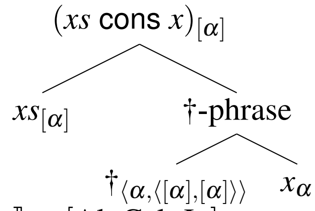
Here I propose that *and* and *or* are essentially data structure markers of lists in the natural language English. Their syntactic and semantic contribution includes three parts: (i) a list-building operator, which adds an item into a list, (ii) a (non)deterministic operator, which filters the list, and (iii) a fold operator, which makes it possible to semantically flatten the list so that the list can further compose with the rest of the sentence.

3.1 List-building operator † (i.e., cons)

I write the list-building operator as † and consider it to be a natural language counterpart of the programming language operator cons. As (10) shows, this infix operator † takes (i) an argument *x* of type α at its right and (ii) a non-empty list argument *xs* of type $[\alpha]$ at its left, and returns a list – *xs* cons *x* – of type $[\alpha]$. To begin with, I assume that any lexical item *x*'s meaning $[\![x]\!]$ could be either *x* (of type α) or a singleton list $[x]$ (of type $[\alpha]$).

(10) a. $[\![\dagger]\!] = \mathsf{cons}_{\langle\alpha,\langle[\alpha],[\alpha]\rangle\rangle} =_{def} \lambda x_\alpha \lambda xs_{[\alpha]}.(xs\ \mathsf{cons}\ x)$
$\rightsquigarrow [\![\dagger Z]\!] = \lambda xs.(xs\ \mathsf{cons}\ Z)$. Thus, $[\![[X_1,...X_n]\ \dagger\ Z]\!] = ([\![\dagger Z]\!])[X_1,...X_n]$

b. The LF (based on Munn (1993)):


$(xs\ \mathsf{cons}\ x)_{[\alpha]}$
$xs_{[\alpha]}$
†-phrase
$\dagger_{\langle\alpha,\langle[\alpha],[\alpha]\rangle\rangle}$
x_α

c. E.g., $[\![[\text{Al}]\dagger\text{Cal}]\!] = [\text{Al, Cal}]$; $[\![[\text{Al, Cal}]\dagger\text{Jo}]\!] = [\text{Al, Cal, Jo}]$

As (10b) shows, I follow Munn (1993) in analyzing the †-phrase as an adjunct to the list *xs*. Essentially, this operator † adds a new item at the right side of the original list. In using †, the asymmetry between the types of its two arguments – α and $[\alpha]$ – guarantees that the building of a MC construction goes one by one recursively from left to right, and the last instance of † occurs before the rightmost item. Therefore, this infix operator † implements the syntactic asymmetry in using *and* and *or*, and syntactically, MC is built in a nested way.

3.2 Operators $\overline{\wedge}$ and $\underline{\vee}$: an identity function and a choice function variable

I write the two list-filtering operators as $\overline{\wedge}$ and $\underline{\vee}$: they take a list *xs* of type $[\alpha]$ and return a list of the same type. $\overline{\wedge}$ keeps the whole list unchanged and $\underline{\vee}$ generates a sublist.

In fact, $\overline{\wedge}$ works as an identity function, which is deterministic and keeps everything in a list – the items and the order among the items – unchanged, as (11) shows.

(11) a. $[\![\overline{\wedge}]\!]_{\langle[\alpha],[\alpha]\rangle} =_{def} \lambda xs_{[\alpha]}.xs$

b. E.g., $\overline{\wedge}$[Nadal, Federer, Nadal] = [Nadal, Federer, Nadal]

In contrast, the operator $\underline{\vee}$ is nondeterministic. As (12a) shows, $\underline{\vee}$ takes a non-emtpy list *xs* as its argument, working like a choice function variable over the set containing all the non-emtpy sublists of *xs*, and returns a non-emtpy sublist of *xs*. I assume this choice function variable either gets its value from the context or is existentially closed, and we can simultaneously trace all the possible results of its application, as shown in (12b).

(12) a. $[\![\veebar]\!]_{\langle[\alpha],[\alpha]\rangle} =_{def} \lambda xs_{[\alpha]}.f_{\text{choice}}(\wp(xs))$
here $\wp([x]) = \{[x]\}$; $\wp(xs \text{ cons } x) = \wp(xs) \cup \{ys \text{ cons } x \mid ys \in \wp(xs) \cup \{[\,]\}\}$

b. E.g., $\begin{cases} \veebar[\text{A, B, C}] = [\text{A}] & \veebar[\text{A, B, C}] = [\text{B}] & \veebar[\text{A, B, C}] = [\text{A, B}] \\ \veebar[\text{A, B, C}] = [\text{C}] & \veebar[\text{A, B, C}] = [\text{A, C}] & \veebar[\text{A, B, C}] = [\text{B, C}] \\ \veebar[\text{A, B, C}] = [\text{A, B, C}] \end{cases}$

The application of $\veebar$ will potentially lose the order information of the original list, i.e., the *n*-th element in the list *xs* is not necessarily the same as the *n*-th element in the list $\veebar(xs)$. This explains the fact that order-sensitivity is not shown in using *or*: e.g., while *and* is often used along with such words like *respectively*, *or* is not compatible with them.

3.3 Fold operator $\oint$ (i.e., fold)

I write the fold operator as $\oint$ and consider it to be a natural language operator working in the same way as the programming language operator fold. As (13) shows,[4] $\oint$ takes a list *xs* of type $[\alpha]$ and returns a partially applied function – $\lambda g.\text{fold } g\ xs$: this is a function that takes a function *g* of type $\langle\alpha,\alpha\alpha\rangle$ (either from the linguistic context or from the world knowledge) and returns an output of type α; the output can be defined in an inductive way.

(13) $[\![\oint]\!] = \text{fold}_{\langle[\alpha],\langle\langle\alpha,\alpha\alpha\rangle,\alpha\rangle\rangle} =_{def} \lambda xs_{[\alpha]}.(\lambda g_{\langle\alpha,\alpha\alpha\rangle}.\text{fold } g\ xs)$
in which fold $g\ [\,]$ is undefined; fold $g\ [x] = x$; fold $g\ (xs \text{ cons } x) = g\ (\text{fold } g\ xs)\ x$

The application of $\oint$ on a list makes the list ready to be flattened, and thus $\oint$ implements the semantic symmetry in using *and/or*. However, the actual flattening only takes place when a function argument *g* combines with the partially applied function $\lambda g.\text{fold } g\ xs$, as illustrated in (14): here the essential meaning of *Al and Jo* remains unambiguous itself; it is the linguistic context – the meaning postulate of the predicate – that determines how *Al and Jo* is actually interpreted, collectively or distributively. In fact, once *g* combines with $\lambda g.\text{fold } g\ xs$, the order information of *xs* gets wiped out; §4.2 will show that this combination can and needs to be delayed in order-sensitive data such as *respectively* sentences.

(14) a. $[\![\text{Al and Jo}]\!] = \oint[\text{Al, Jo}] = \lambda g.\text{fold } g\ [\text{Al, Jo}]$
b. Al and Jo met. $\forall e.\text{meet}(e) \to \text{Non-atom}(\text{agent}(e)) \rightsquigarrow g = \oplus_{\langle e,ee\rangle}$
c. Al and Jo smiled. $\forall e.\text{smile}(e) \to \text{Atom}(\text{agent}(e)) \rightsquigarrow g = \sqcap_{\langle\alpha,\alpha\alpha\rangle}$

[4]The fold in (13) is a left fold. In fact, (13) is a special case of the more general implementation – $\text{foldl}_{\langle[\alpha],\langle\langle\beta,\alpha\beta\rangle,\beta\beta\rangle\rangle} =_{def} \lambda xs_{[\alpha]}.(\lambda g_{\langle\beta,\alpha\beta\rangle}\lambda z_\beta.\text{foldl } g\ z\ xs)$, in which foldl $g\ z\ [\,] = z$; foldl $g\ z\ (xs \text{ cons } x) = g\ (\text{foldl } g\ z\ xs)\ x$ (here foldl is also a left fold). Whether and how natural language empirically (i) distinguishes the left and the right folds and (ii) motivates the more general implementation are left for future studies.

3.4 Pronounced *and* and *or* are compositions of three operators

The operator definitions given in §3.1-§3.3 have shown that (i) $\dagger$ and $\int$ implement respectively the syntactic asymmetry and the semantic symmetry in using coordinators, (ii) $\overline{\wedge}$ keeps the order-sensitivity in using *and*, and (iii) $\underline{\vee}$ implements the nondeterminism in using *or*. As (15) shows, here I propose that (i) all these operators (i.e., $\dagger$, $\overline{\wedge}$, $\underline{\vee}$ and $\int$) are silent themselves, and (ii) pronounced *and* and *or* are function compositions of three operators, and get pronounced where the last application of the infix operator $\dagger$ happens. Evidently, the current proposal accounts for all the crucial syntactic and semantic properties of pronounced *and* and or: essentially, they mark the boundary of a list.

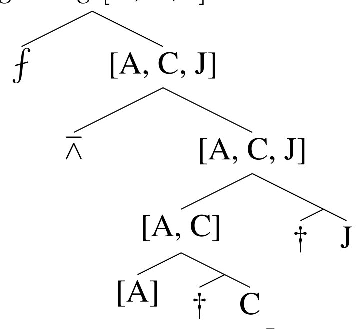
(15) a. $[\![\text{and } x]\!] =_{def} \lambda xs.\int(\overline{\wedge}(xs \dagger x)) = \lambda xs.\int(\overline{\wedge}((\dagger x)(xs))) = \int.\overline{\wedge}.(\dagger\, x)$
b. $[\![\text{or } x]\!] =_{def} \lambda xs.\int(\underline{\vee}(xs \dagger x)) = \lambda xs.\int(\underline{\vee}((\dagger x)(xs))) = \int.\underline{\vee}.(\dagger\, x)$

4. Accounting for multiple coordination and order-sensitivity

4.1 The analysis of multiple coordination

(16) and (17) illustrate how the current proposal accounts for multiple coordination.[5]

(16) a. Al, Cal and Jo smiled.
b. $[\![\text{Al, Cal and Jo}]\!] =$
$\int\ (\overline{\wedge}\ [\text{Al, Cal, Jo}]) =$
$\lambda g.\mathsf{fold}\ g\ [\text{A, C, J}]$



c. $[\![\text{Al, Cal and Jo smiled}]\!] =$
$((\lambda g.\mathsf{fold}\, g\, [\text{A, C, J}])\sqcap)\lambda x.\text{smile}(x) =$
$(\lambda P.\text{A}(P)\sqcap\text{C}(P)\sqcap\text{J}(P))\lambda x.\text{smile}(x) =$
$\text{A(smile)} \wedge \text{C(smile)} \wedge \text{J(smile)}$

(17) a. Al, Cal or Jo smiled.
b. $[\![\text{Al, Cal or Jo}]\!] =$
$\int\ (\underline{\vee}\ [\text{Al, Cal, Jo}]) =$
$$\begin{cases} \lambda g.\mathsf{fold}\ g\ [\text{A}] \\ \ldots \\ \lambda g.\mathsf{fold}\ g\ [\text{C, J}] \\ \lambda g.\mathsf{fold}\ g\ [\text{A, C, J}] \end{cases}$$
c. $[\![\text{Al, Cal or Jo smiled}]\!] =$
$$\begin{cases} (\mathsf{fold}\ \sqcap\ [\text{A}])\lambda x.\text{sml}(x) \\ \ldots \\ (\mathsf{fold}\ \sqcap\ [\text{A, C, J}])\lambda x.\text{sml}(x) \end{cases}$$
$$= \begin{cases} \text{A(sml)} \\ \ldots \\ \text{A(sml)} \wedge \text{C(sml)} \wedge \text{J(sml)} \end{cases}$$

(16b) shows that in building a MC construction, only the silent operator $\dagger$ is recursively used. *and* and *or* are only pronounced when other operators – $\overline{\wedge}$/$\underline{\vee}$ and $\int$ – come to work.

This means that, in the data (6), $\overline{\wedge}$ and $\int$ appear twice in (6a), but only once in (6b): in (6a), to generate the '2 requirements' reading, fold is used twice, first with a contextually salient $\oplus$ to sum up two events – dance_e and jump_e – and then with a $\sqcap$; in using $\sqcap$, the event sing_e and the big event – $(\text{dance}\oplus\text{jump})_e$ – are semantically on the same level.

[5]I assume the type of names (e.g., *Al*) can be either *e* or $\langle et,t\rangle$; (17) shows all potentially possible results.

For (6a), if both folds take a $\sqcap$ as its argument, the '3 requirements' reading can also be derived. However, the current theory suggests that every instance of *and*/*or* marks a ready-to-be-fold list, the items in which must be on the same level semantically. Thus, for the '3 requirements' reading, there have to be two contextually salient levels in using (6a): dance and jump on one level; sing and ($\mathsf{fold}\ \sqcap$ [dance, jump]) on the other. If sing, dance and jump are all on the one same level in a context, (6b) is predicted to be preferred for the '3 requirements' reading. Similarly, the MC pattern of (9) is acceptable only if in a context, A and B as a whole are on the same level as each of the other individuals C, D, and E.

Notice that in (17) it is $\veebar$ that implements the nondeterminism in using *or*, and the function g used along with fold in (17) is $\sqcap$, not $\sqcup$. Then in using *or*, can $\oplus$ be the function g to combine with $\lambda g.\mathsf{fold}\ g\ xs$? The current analysis predicts that this is possible.

Cross-linguistically, words corresponding to English *and* is compatible with both distributive and collective predicates (e.g., *A and B smiled*; *A and B met*), while words corresponding to English *or* is incompatible with collective predicates (e.g., * *A or B met*). This empirical generalization seems to be against our prediction, but here I argue that this is not the case – the current analysis can still account for the ungrammaticality of *A or B met*: as a collective predicate, *meet* requires its agent to be a plural entity; however, as shown in (12), using $\veebar$ can potentially yield a singleton list, and thus the result of $(\digamma\,.\veebar.[...])\ \oplus$ might be an atom, failing to guarantee that the requirement of *meet* be satisfied. As far as the satisfaction of this requirement of collective predicates can be guaranteed, as (18) shows,[6] it is empirically motivated and conceptually elegant to use $\oplus$ along with $\digamma\,.\veebar.[...]$.

(18) I heard [John and [Mary or Sue]] built a raft together.
$[\![$Mary or Sue$]\!]_\oplus = (\digamma(\veebar[\text{M, S}]))\oplus$ All possible outputs: M, S, M $\oplus$ S
$[\![$[John and [Mary or Sue]]$]\!]_\oplus$ All possible outputs: J $\oplus$ M, J $\oplus$ S, J $\oplus$ M $\oplus$ S

4.2 The analysis of order sensitivity: *respectively* sentences

(19) Al and Cal married Jo and Sue respectively. $\rightsquigarrow$ A married J, and C married S.

(19) shows that *respectively* expresses an order parallelism between two lists: the order information of each list needs to be kept until the two lists are zipped up. To account for (19), I adopt two more operators (type-shifters) from functional programming languages:[7]

(20) a. Type of fmap: $(a \to b) \to f\ a \to f\ b$ (f is a type constructor)
$\underline{f \text{ is } [\,]}$: $\mathsf{fmap}\ k_{a\to b}\ [\,] = [\,]$; $\mathsf{fmap}\ k_{a\to b}\ (xs\ \mathsf{cons}\ x)_{[a]} = (\mathsf{fmap}\ k\ xs)\ \mathsf{cons}\ (k\ x)$
$\underline{f \text{ is } (\lambda x_r.\ ...)_{r\to}}$: $\mathsf{fmap}\ k_{a\to b}\ g_{r\to a} = \lambda x_r.k(g(x))$

b. Type of $(<\!*\!>)$: $f\ (a \to b) \to f\ a \to f\ b$ (f is a type constructor)
$\underline{f \text{ is } [\,]}$: $(<\!*\!>)\ xs\ [\,] = [\,]$; $(<\!*\!>)\ [\,]\ ys = [\,]$;
$(<\!*\!>)\ (xs\ \mathsf{cons}\ x)_{[a\to b]}(ys\ \mathsf{cons}\ y)_{[a]} = ((<\!*\!>)\ xs\ ys)\ \mathsf{cons}\ (x\ y)$
$\underline{f \text{ is } (\lambda x_r.\ ...)_{r\to}}$: $(<\!*\!>)\ k_{r\to(a\to b)}\ g_{r\to a} = \lambda x_r.k\ x\ (g\ x)$

[6] I thank Manuel Križ (p.c.) for pointing this out to me.
[7] See Charlow (2014) and Bumford (2015) for more uses of similar tools. Here $\langle ab\rangle$ is written as $a \to b$.

(20) shows the types of the two operators – fmap and $(<\!*\!>)$ – as well as their definitions for two type constructors – [ ] and $(\lambda x_r.\ ...)_{r\rightarrow}$. A type constructor f (e.g., [ ]) takes a concrete type (e.g., α) to build a new concrete type (e.g., $[\alpha]$ – a concrete list type); in the current case, f is a composition of $\lambda g.\mathsf{fold}\ g$ ... and [ ]: $\lambda g.\mathsf{fold}\ g$ [ ]. The meaning of *respectively* can be considered as a special case of $(<\!*\!>)$: it requires that the constructor f can record order information, such as [ ] and our constructor $\lambda g.\mathsf{fold}\ g$ [ ]; *respectively* zips up the values according to their order. As a result, (21) is a compositional analysis for (19):

(21) a. $[\![\text{married}]\!]_{\langle e,et\rangle} =_{def} \lambda z\lambda x.\text{married}(z)(x)$ $\quad\rightsquigarrow x$ married z

b. $[\![\text{Jo and Sue}]\!] = ⨍(\overline{\wedge}[\text{Jo, Sue}]) = \lambda g.\mathsf{fold}\ g\ [\text{Jo, Sue}]$

$[\![\text{Al and Cal}]\!] = ⨍(\overline{\wedge}[\text{Al, Cal}]) = \lambda g.\mathsf{fold}\ g\ [\text{Al, Cal}]$

c. $\mathsf{fmap}\ [\![\text{married}]\!]\ [\![\text{Jo and Sue}]\!]$

$= \lambda g.\mathsf{fold}\ g\ [\lambda x.\text{married(Jo)}(x), \lambda x.\text{married(Sue)}(x)]$

d. $[\![\text{respectively}]\!]([\![\text{Al and Cal}]\!](\mathsf{fmap}\ [\![\text{married}]\!]\ [\![\text{Jo and Sue}]\!]))$

$= (<\!*\!>)([\![\text{Al and Cal}]\!](\mathsf{fmap}\ [\![\text{married}]\!]\ [\![\text{Jo and Sue}]\!]))$

$= \lambda g.\mathsf{fold}\ g\ [\text{married(Jo)(Al)}, \text{married(Sue)(Cal)}]$

e. When $g = \wedge$, $[\![(19)]\!] = \text{married(Jo)(Al)} \wedge \text{married(Sue)(Cal)}$

As (21e) shows, while the use of pronounced *and* marks the end of building a list and makes the list ready for further computations (in this sense, $⨍$ is just a type-shifter bridging a list and other parts of a computation), it is the application of the function g that eventually ends the order-sensitivity of a list. Such a division of labor between *and* X (i.e., $⨍\,.\,\overline{\wedge}\,.(\dagger X)$) and g shows up clearly in *respectively* sentences, in which the application of g is delayed, so that the order information of the two lists involved is kept until they are zipped up.

5. Summary and outlook

In this paper, I argue for a decompositional analysis to account for the syntactic and semantic contribution of natural language particles *and* and *or*: although morphologically simple, each of these two particles works as a list marker implementing three basic operations, namely (i) building a list, (ii) handling the (non)determinism and (iii) delivering the list for further computations. Crucially, the current analysis argues that such operations as set intersection (i.e., $\sqcap$) and group forming (i.e., $\oplus$) are not performed by *and* itself.

Thus the current work is consistent with what is advocated in Winter (1995, 1998) and Szabolcsi (2015). There are two interesting follow-up questions:

(i) Does natural language overtly express such operations as $\sqcap$ or $\oplus$ at all? Notice that *red boat* certainly means $\lambda x.\text{red}(x) \sqcap \text{boat}(x)$, but it would be really surprising if a language overtly expresses $\sqcap$ here. If $\sqcap$ and $\oplus$ are never overtly expressed, then what triggers such a meaning composition and how is the complex meaning processed by our brain?

(ii) Japanese *mo* (see Szabolcsi (2015)) and English *and* represent two kinds of particles involved in conjunction expressions, and they have very different distributions. Both the current analysis and Szabolcsi (2015) argue that in fact, they do not perform intersection themselves. Then can we further have a unified account for them? A potential promising way might be to find out the basic operators underlying all these overt particles.

References

Barker, Chris, & Chung-chieh Shan. 2014. *Continuations and Natural Language*. Oxford University Press.

Bumford, Dylan. 2015. Incremental quantification and the dynamics of pair-list phenomena. *Semantics and Pragmatics* 8:1 – 70.

Champollion, Lucas. 2015. Ten men and women got married today: noun coordination and the intersective theory of conjunction. *Journal of Semantics* (in press).

Charlow, Simon. 2014. On the semantics of exceptional scope. Doctoral dissertation, New York University.

Florio, Salvatore, & David Nicolas. 2014. Plural logic and sensitivity to order. *Australasian Journal of Philosophy*. DOI: 10.1080/00048402.2014.963133.

Gazdar, Gerald. 1980. A cross-categorial semantics for coordination. *Linguistics and Philosophy* 3:407 – 409.

Heycock, Caroline, & Roberto Zamparelli. 2005. Friends and colleagues: plurality, coordination, and the structure of DP. *Natural Language Semantics* 13:201 – 270.

Krifka, Manfred. 1990. Boolean and non-boolean 'and'. In *Papers from the 2nd Symposium on Logic and Language*, ed. László Kálmán & László Pólos, 161 – 188. Hungary: Akadémiai Kiadó.

Lasersohn, Peter. 1995. *Plurality, conjunction and events*. Dordrecht, Netherlands: Kluwer. Studies in Linguistics and Philosophy (vol. 55).

Montague, Richard. 1973. The proper treatment of quantification in ordinary English. In *Approches to Natural Language*, ed. K. J. J. Hintikka, J. M. E. Moravcsik, & P. Suppes, 221 – 242. Dordrecht: D. Reidel.

Munn, Alan Boag. 1993. Topics in the Syntax and Semantics of Coordinate Structures. Doctoral dissertation, University of Maryland.

Partee, Barbara, & Mats Rooth. 1983. Generalized conjunction and type ambiguity. In *Meaning, use and interpretation of language*, 361 – 383. Germany, Berlin: de Gruyter.

von Stechow, Arnim. 1974. ε-λ kontextfreie Sprachen: Ein Beitrag zu einer naturlichen formalen Semantik. *Linguistische Berichte* 34:1 – 33.

Szabolcsi, Anna. 2015. What do quantifier particles do? *Linguistics and Philosophy* 38:159 – 204.

Tarski, Alfred. 1935. Zur Grundlegung der Booleschen Algebra. *Fundamenta Mathematicae* 24:177 – 198.

Winter, Yoad. 1995. Syncategorematic conjunction and structured meanings. In *Proceedings of Semantics and Linguistic Theory, SALT5*, ed. Mandy Simons & Teresa Galloway. Cornell University, Ithaca, NY: CLC Publications.

Winter, Yoad. 1998. Flexible Boolean Semantics: coordination, plurality and scope in natural language. Doctoral dissertation, Utrecht University.

Winter, Yoad. 2006. Multiple coordination: meaning composition vs. the syntax-semantics interface. Unpublished manuscript.

Linmin Zhang
linmin.zhang@nyu.edu