Decomposing English *and* and *or*

Linmin Zhang (linmin.zhang@nyu.edu)
Department of Linguistics, New York University

Summary

- Using **silent operators** to **decompose English particles *and* and *or***:
  - (1) a. List-building operator †
    b. List-filtering operators:
    i. identity function operator $\overline{\wedge}$; ii. choice function variable $\veebar$
    c. Fold operator f
- 'f . $\overline{\wedge}$.†' is realized as ***and***; 'f . $\veebar$.†' is realized as ***or***.
- *and* / *or*: **not** $\sqcap$/ $\oplus$ /$\sqcup$ operators per se, but **data structure markers**.

Motivations for introducing the list type

Replication and order-sensitivity

- The items connected by *and* could be the same.
- Sometimes the order between the items affects the truth condition.
  - (2) By definition, the first two numbers in the Fibonacci sequence are **1 and 1**, or **0 and 1**.
  - (3) The last three champions at Roland Garros were **Nadal, Federer and Nadal**. (Florio & Nicholas, 2014)

How to account for multiple coordination (MC)? (see Winter 2006)

- (4) a. ✓ A, B **and** C came.
  b. ✓ A **and** B **and** C came.
- (5) a. * A **and** B, C came.
  b. * A, B, C came.
- **Hyp.1** *and* and *or* **(i) are binary operators** and **(ii) could be silent**.
  - *and* and *or* are **recursively** used in MC.
  - MC is built in a **nested** way: e.g., [[*A ~~and~~ B*] *and C*].
  - **stipulation**: *and*/*or* can freely go silent except in its last application.
- **Problem 1** as far as the types of *and* / *or* are of the form $\langle \tau, \tau\tau \rangle$, such a stipulation cannot help over-generating ill-formed expressions:
  - (6) * [[Al ~~and~~ Bill] **and** [Cal ~~and~~ Jo]] smiled.
- **Problem 2** silent *and* would also over-generate readings (Winter 2006):
  - (7) a. You need to sing and dance and stamp your feet.
    ✓ 2 requirements (sing; dance and stamp); ✓ 3 requirements
    b. You need to sing, dance and stamp your feet.
    # 2 requirements; ✓ 3 requirements
- **Hyp.2** *and* and *or* **(i) are never silent** and **(ii) could be *n*-ary**.
  - *and* and *or* can be generalized as ***n*-ary operators**.
  - MC is built in a **flat** way.
  - **stipulation**: *and* and *or* are always pronounced, between the rightmost two items.
  - (8) a. $\sqcap^n_{\tau^1(\tau^2 \ldots (\tau^n \tau))} = \lambda X^1_\tau \ldots \lambda X^n_\tau .\, \sqcap^2 (X^1, \sqcap^{n-1}(X^2, \ldots, X^n))$
    b. $\sqcup^n_{\tau^1(\tau^2 \ldots (\tau^n \tau))} = \lambda X^1_\tau \ldots \lambda X^n_\tau .\, \sqcup^2 (X^1, \sqcup^{n-1}(X^2, \ldots, X^n))$
- **Problem 1 still exists** – there would be over-generations:
  - (9) * [A, [B **and**^{2} C], D, E **and**^{5} F] smiled.
    – could be genarated via the use of a quinary $\sqcap^5$ plus a binary $\sqcap^2$
    (i.e., There is no constraint forcing the use of a 6-ary operator.)

Details of the decompositional proposal

List-building operator †

- The **infix** operator † takes **(i)** a non-empty list argument *xs* of type $[\alpha]$ **at its left** and **(ii)** an argument *x* of type α **at its right**, and returns a list (*xs* : *x*) of type $[\alpha]$ (see also Charlow 2014).
  - (10) $[\![\dagger]\!] =_{def} \lambda xs_{[\alpha]} \lambda x_\alpha .(xs : x)$
- The **asymmetry** between the types $[\alpha]$ and α in using † guarantees that the building of MC goes **one by one recursively from left to right**, and the last instance of † inserts between the two rightmost items.
- To begin with, I assume that any lexical item's meaning $[\![x]\!]$ could be either x_α or a singleton list $[x]_{[\alpha]}$.
- This infix operator † implements Winter 2006's idea: **syntactically, MC is built in a nested way**.

$[\![\text{John, Mary and Bill}]\!] =$
$f\ [\overline{\wedge}\ [\ [\text{John} \dagger \text{Mary}] \dagger \text{Bill}\ ]\ ] =$
$\lambda g \lambda z.\text{fold}\ [\text{John, Mary, Bill}]\ g\ z$
$[\![\text{John, Mary or Bill}]\!] =$
$f\ [\veebar\ [\ [\text{John} \dagger \text{Mary}] \dagger \text{Bill}\ ]\ ] =$

$$f \begin{cases} [\text{J}] & = \lambda g \lambda z.\text{fold}\ [\text{J}]\ g\ z \\ [\text{M}] & = \lambda g \lambda z.\text{fold}\ [\text{M}]\ g\ z \\ \ldots & \\ [\text{M, B}] & = \lambda g \lambda z.\text{fold}\ [\text{M, B}]\ g\ z \\ [\text{J, M, B}] & = \lambda g \lambda z.\text{fold}\ [\text{J, M, B}]\ g\ z \end{cases}$$

List-filtering operators $\overline{\wedge}$ and $\veebar$

- $\overline{\wedge}$ and $\veebar$ take a list *xs* of type $[\alpha]$ and return a list of type $[\alpha]$.
- Essentially, $\overline{\wedge}$ is an **identity function operator**, which is **deterministic** and keeps the order among items within a list.
- $\veebar$ is a **choice function variable** over the power set containing sets composed from items within the list – the order within the original list would be lost.
  - (11) a. $[\![\overline{\wedge}]\!] =_{def} \lambda xs_{[\alpha]}.xs$
    b. $[\![\veebar]\!] =_{def} \lambda xs_{[\alpha]}.ys$
    $(ys \neq [\,] \wedge ys \in \wp(xs))$
- Since $\veebar$ is a choice function variable, it would get its value from the context or be existentially closed.

Fold operator f (here it's a left fold.)

- f takes **(i)** a list *xs* of type $[\alpha]$ and **(ii)** a function *g* of type $\langle \beta, \alpha\beta \rangle$ and a starting point *z* (of type β) corresponding to the function argument *g*, and returns a value of type β (see also Bumford 2014).
  - (12) $[\![f]\!] =_{def} \lambda xs_{[\alpha]} \lambda g_{\langle \beta, \alpha\beta \rangle} \lambda z_\beta .\text{fold}\ xs\ g\ z$
- The output is defined in an inductive way:
  - (13) fold [ ] g $z = z$
    fold (xs : x) g $z = g$ (fold xs g z) x
- In applying f, the function argument *g* and the corresponding starting point *z* **come from the context or other parts of the sentence**.
- Here are two examples:
  - $g = \sqcap_{\langle \alpha, \alpha\alpha \rangle}$, $z = \text{True}_t$ (or $\mathbb{U}_{\langle \sigma_1, \sigma_2 \rangle}$)
  - $g = \oplus_{\langle e, ee \rangle}$, $z = \varnothing_e$
- The fold operator f implements Winter 2006's idea: **semantically, MC is interpreted in a flat way**.
- Only † is recursively used in building an MC construction; *and/or* is pronounced only when $\overline{\wedge}$/$\veebar$ and f come to work.
- This means that $\overline{\wedge}$ and f appear twice in (7a), but only once in (7b): In (7a), a contextually salient *g* (a sum operator for events, see Lasersohn 1995) becomes the argument of the first f and turns *dance and stamp your feet* into a sum in the '2 requirements' reading.
- If both *g* are $\sqcap$, the '3 requirements' reading of (7a) could also be generated – however, since *g* comes from the context, if the context doesn't provide two salient $\sqcap$, the current theory would predict that (7b) is the preferred one for the '3 songs' reading.

Let's look at some other data!

The function *g* used in *between ... and ...*

- The current analysis suggests that some linguistic contexts might provide a *g* other than $\sqcap$ and $\oplus$.
  - (14) The relationships between John, Bill and Peter, and Mary, Sue and Katie were all short.
  - (15) There are 4 prime numbers between 1 and 8.
  - (16) She is busy between NY, London and Paris.
- *between* might provide a *g* to generate all possible pairs under the context.

The use of $\oplus$ in *A, B or C*: is it possible?

- Such a collective predicate as *built a raft together* requires its agent to be non-singular; however, $\veebar$ might potentially return a singleton list, which could not guarantee that the requirement of the collective predicate be satisfied.
  - (17) * John, Mary or Bill built a raft together. 3.3/7
    (cf., * *John built a raft together*. 1.6/7)
- However, the use of $\oplus$ might still be possible sometimes:
  - (18) I saw John and Mary or Sue kissed.

Selected references **Bumford. 2014.** Incremental quantication and the dynamics of pair-list phenomena. Ms in review. | **Charlow. 2014.** On the semantics of exceptional scope. NYU Dissertation. | **Winter. 2006.** Multiple coordination: meaning composition vs. the syntax-semantics interface. Ms. **Acknowledgements** Lots of thanks to Lucas Champollion and Yoad Winter for suggestions and discussions through various stages of this project! I also thank Chris Barker, Dylan Bumford, Simon Charlow, Ivano Ciardelli, Kit Fine, Sonia Kasyanenko, Manuel Križ, Jeremy Kuhn, Hoi Ki Law, David Nicolas, Jim Pryor, Philippe Schlenker, Anna Szabolcsi, Adina Williams, Yimei Xiang and the reviewers of NELS45 for comments.