A Quick Refresher on the Semantics of NPs and VPs

The following ideas are probably old hat to you now. However, let’s quickly refresh our memory on certain key ideas regarding the semantics of NPs and VPs.

1. The Semantics of NPs

The extension of an NP can be thought of as an <e,t> function.

1.1 Extension of NP as <e,t> Function

\[
[[ \text{boy} ]] = [ \lambda x . \ x \text{ is a boy } ] = \{ <\text{Bill}, T>, <\text{Sue}, F>, <\text{Joe}, T>, <\text{Joy}, F> \}
\]

Type <e,t> functions are equivalent to sets of individual entities (namely, the set of entities the function assigns ‘T’ to). Thus, we can also think of the extension of an NP as a set of entities.

1.2 Extension of NP as a Set of Entities

\[
[[ \text{boy} ]] = \{ \text{Bill}, \text{Joe} \}
\]

2. Classic Semantics for VPs and Vs

2.1 Classic Semantics for VPs and Intransitive Vs

The extension of a VP and an intransitive V can be thought of as an <e,t> function.

2.1.1 Extension of VP and Intransitive V as <e,t> Function

\[
a. \quad [[ \text{kissed Sue} ]] = [ \lambda x . \ x \text{ kissed Sue } ] = \{ <\text{Bill}, T>, <\text{Sue}, F>, <\text{Joe}, F>, <\text{Joy}, T> \}
b. \quad [[ \text{danced} ]] = [ \lambda x . \ x \text{ danced } ] = \{ <\text{Bill}, T>, <\text{Sue}, T>, <\text{Joe}, F>, <\text{Joy}, F> \}
\]

Thus, the extension of a VP or intransitive V can also be thought of as a set of individual entities.

2.1.2 Extension of VP and Intransitive V as a Set of Entities

\[
a. \quad [[ \text{kissed Sue} ]] = \{ \text{Bill}, \text{Joy} \}
b. \quad [[ \text{danced} ]] = \{ \text{Bill}, \text{Sue} \}
\]

1 For further reading complementing these notes, I refer you to Kratzer (2002: Chapter 1), available at: http://www.semanticsarchive.net/Archive/GU1NWM4Z/
2.2 Classic Semantics for Transitive Vs

The extension of a transitive V can be thought of as an \<eet> function. This is a function from entities to functions from entities to T-values.

(5) **Extension of Transitive V as \<eet> Function**

\[
[[ \text{kissed} ]] \quad = \quad [ \lambda y. [ \lambda x . x \text{kissed} y ] ]
\]

\[
= \quad \{ < \text{Bill} , \{ < \text{Sue}, \text{T}> , < \text{Tom}, \text{F} > \ldots \} > , \quad < \text{Tom} , \{ < \text{Sue}, \text{F}, < \text{Bill}, \text{T} > , \ldots \} > \ldots \}
\]

Such \<eet> functions are equivalent to *sets of pairs of individuals* (namely, the set of pairs \<xy> such that the function takes y and returns a function that takes x and returns T).

(6) **Extension of Transitive V as Set of Pairs of Entities**

\[
[[ \text{kissed} ]] \quad = \quad \{ < \text{Bill}, \text{Sue}>, < \text{Bill}, \text{Frank}>, < \text{Sue}, \text{Bill}>, < \text{Frank}, \text{Bill} > \}
\]

*This is the extension of ‘kissed’ in a world/time where Bill and Sue kiss and Bill and Frank kiss, and nobody else kisses.*

---

3. Event-Based Semantics for VPs and Vs

All of this is pretty boring so far, right? The reason for mentioning it, though, is to lay the groundwork for a quick refresher on event-based semantics for Vs and VPs.

3.1 The Event-Based Semantics for VPs

If you ask someone off the street what the sentence *Dave danced* ‘refers’ to, you might get the intuitive answer that it refers to some specific *event* (or *situation*). For the past 30 years, semanticists have been developing this rough, intuitive notion into a sophisticated theory of verbal predication.

Under such ‘event semantics’ (or ‘situation semantics’), the extension of a VP or an intransitive V can be thought of as function from entities to *functions from events to T-values*. If we suppose that the type for events is s, such functions are of type \<e, <s, t>>.

(7) **Extension of VP as \<e, <s, t>> Function**

\[
[[ \text{kissed Sue} ]] \quad = \quad [ \lambda x . [ \lambda e . \text{e is an event of kissing Sue done by x} ] ]
\]

If we add some logical notation to our metalanguage, we can rewrite the extension in (7) as in (8) below. Such a logical form is often referred to as ‘Neo-Davidsonian’.
(8) **Extension of VP as \( <e, <s, t>> \) Function**

\[
[[ \text{kissed Sue} ]] = [[ \lambda x . [ \lambda e . \text{kissing(e) & Theme(e) = Sue & Agent(e) = x} ]] ]
\]

‘\( e \) is an event of kissing whose theme is Sue and whose agent is \( x \)’

This semantics would also hold for intransitive verbs like ‘danced’.

(9) **Extension of Intransitive V as \( <e, <s, t>> \) Function**

\[
[[ \text{danced} ]] = [[ \lambda x . [ \lambda e . \text{e is an event of dancing done by x} ]] ]
\]

\[
= [[ \lambda x . [ \lambda e . \text{dancing(e) & Agent(e) = x} ]] ]
\]

Just as \( <eet> \) functions are equivalent to sets of pairs of entities, \( <est> \) functions are equivalent to sets of pairs consisting of *entities and events*. Thus, we can also think of the extensions of VPs and intransitive Vs as such sets.

(10) **Extension of VPs and Intransitive Vs as Sets of Pairs of Entities and Events**

a. \[
[[ \text{kissed Sue} ]] = \{ <\text{Bill}, e_1>, <\text{Joy}, e_2> \}
\]

*The extension of ‘kissed Sue’ in a world/time where Bill kissed Sue and Joy kissed Sue.*

b. \[
[[ \text{danced} ]] = \{ <\text{Bill}, e_1>, <\text{Sue}, e_2> \}
\]

*The extension of ‘danced’ in a world/time where Bill danced and Sue danced.*

### 3.1.1 The Semantics of Full Sentences

Under the semantics for VPs and Vs laid out above, a sentence like “Bill danced” would come out as denoting a function from events to T-values.

(11) **Semantic Computation of a Sentence**

\[
[[ \text{Bill danced} ]] = [[ \text{danced} ]] (\text{Bill})
\]

\[
= [[ \lambda x . [ \lambda e . \text{dancing(e) & Agent(e) = x} ]] ] (\text{Bill})
\]

\[
= [ \lambda e . \text{dancing(e) & Agent(e) = Bill} ]
\]

*But, how is this property of events the extension of the sentence? Shouldn’t the extension of the sentence be a T-value?*
(12) The Solution: Existential Closure Over Events

a. Step 1: Syntactic Assumption
   Every sentence may contain a phonologically null operator ‘∃e’. Thus, the structure of a sentence like “Bill danced” would be as follows.

   ![Diagram of sentence structure]

   S
   ┌───┐
   │ ∃e │
   └───┘
   S
   ┌──┘
   │ DP
   └──┘
   Bill
   ┌──┘
   │ VP
   └──┘
   danced.

b. Step 2: Semantic Step
   
   \[ \lambda P_{<st>} . \text{There exists an event } e \text{ such that } P(e) = T \]

Putting these parts together, we derive that the meaning of a sentence like (12a) is the following:

(13) Computing the Meaning of a Sentence in Event Semantics

a. \[ [[ \exists e \ [ \text{Bill danced }] ]] = \]

b. \[ [[ \exists e ]] \ ( [[ \text{Bill danced }] ] ) = \]

c. \[ \lambda P_{<st>} . \text{There exists an event } e \text{ such that } P(e) = T \] \( ( [[ \text{Bill danced }] ] ) = \)

d. \text{There exists an event } e \text{ such that } [\text{Bill danced}] (e) = T

e. \text{There exists an event } e \text{ such that } [\lambda e . \text{dancing}(e) \& \text{Agent}(e) = \text{Bill }] (e) = T

f. \text{There exists an event } e \text{ such that } \text{dancing}(e) \& \text{Agent}(e) = \text{Bill.}
   \text{There is an event of dancing that has Bill as its agent.}

(14) Important Unanswered Question

OK, but why view the T-conditions of a sentence like “Bill danced” as being like (13f), making explicit reference to an ‘event’ of dancing (whatever the hell that is?...)

(15) Promissory Note

In previous classes (620), you’ve seen the role that events can play in the semantics of aspect and tense. In this class, you will see that there is some interesting work they do in the semantics of plurals and related problems...
3.2 The Event-Based Semantics for Transitive Vs

Given that the extension of a VP can be thought of as \(<est>\) function, the extension of a transitive verb can be thought of as a function from entities to \(<est>\) functions; i.e., \(<est>\) functions.

(16) Extension of Transitive V as \(<est>\) Function

\[ \text{[[ kissed ]] } = \ [\lambda y. [\lambda x. [\lambda e. e \text{ is an event of kissing } y \text{ done by } x ]]] \]

\[ = \text{Under a ’Neo-Davidsonian’ Representation:} \]

\[ [\lambda y. [\lambda x. [\lambda e. \text{ kissing}(e) \& \text{ Theme}(e) = y \& \text{ Agent}(e) = x ]]] \]

As before, we can also think of the extension of such a transitive V as a particular kind of set. In this case, though, it is a set of triples: the set of triples \(<x,y,z>\) such that the function takes the entity \(z\), and returns a function that takes the entity \(y\), and returns a function that takes the entity \(x\) and returns \(T\).

(17) \[ \text{[[ kissed ]] } = \{ <\text{Bill, Sue, } e_1>, <\text{Bill, Frank, } e_2>, <\text{Sue, Bill, } e_3>, <\text{Frank, Bill, } e_4> \} \]

This is the extension of ‘kissed’ in a world/time where Bill and Sue kiss and Bill and Frank kiss, and nobody else kisses.

4. Kratzer’s Event-Based Semantics for Transitive Vs

Over the years, Angelika has argued for a semantics of transitive Vs where they are relations only between (single) entities and events, that is, where they are functions of type \(<est>\). Under this semantics, the extension of a transitive V like \(\text{kissed}\) is as follows.

(18) Extension of Transitive Vs in Kratzerian Event-Based Semantics

\[ \text{[[ kissed ]] } = \ [\lambda y. [\lambda e. e \text{ is an event of kissing } y ] ] \]

\[ = \text{Under a ‘Neo-Davidsonian’ Representation} \]

\[ [\lambda y. [\lambda e. \text{ kissing}(e) \& \text{ Theme}(e) = y ] ] \]

\[ = \text{Under Kratzer’s Preferred Notation} \]

\[ [\lambda y. [\lambda e. \text{ kiss}(e,y) ] ] \]

As shown below, under this semantics, the extension of a VP like \(\text{kissed Sue}\) would simply be a property of events.
Computation of the Extension of a VP in Kratzerian Semantics

a. \( [[ \text{kissed Sue} ]] = \)

b. \( [[ \text{kissed} ]] (\text{Sue}) = \)

c. \( [\lambda y. [\lambda e. \text{kiss}(e, y)] (\text{Sue}) = \)

d. \( [\lambda e. \text{kiss}(e, \text{Sue})] \)

Crucial Question
How, then, does a VP like \text{kissed Sue} combine semantically with a subject DP like \text{Bill}?

The Answer: Little-v

a. Step 1: Syntactic Assumption

The VP is dominated by a projection vP (‘little vP’). The subject appears in the specifier of ‘little’-v

\[
\begin{array}{c}
\text{DP} \\
\text{vP} \\
\text{vP} \\
\text{vP}
\end{array}
\]

\[
\begin{array}{c}
\text{Bill} \\
\text{v} \\
\text{VP} \\
\text{VP}
\end{array}
\]

b. Step 2: Semantic Assumption

The meaning of the little-v head does the work of introducing the ‘Agent’ role into the semantic representation of the sentence.\(^2\)

\[
[[ v ]] = [\lambda P_{\text{SP}}. [\lambda x. [\lambda e. P(e) \& \text{Agent}(e) = x]]]
\]

With these ideas in place, we can now derive the desired event-based semantics for the sentence \text{Bill kissed Sue}.

\(^2\) The semantics given for little-v in (21) isn’t exactly what appears in Angelika’s papers, but it will suffice for now.
(22) Illustrative Sentence

a. Syntax (at LF, roughly speaking)

\[
S \\
\exists e \quad vP_1 \\
\begin{array}{c}
DP \\
\text{Bill}
\end{array} \\
v \\
vP_2 \\
\begin{array}{c}
V \\
kissed
\end{array} \\
\begin{array}{c}
VP \\
\text{Sue}
\end{array}
\]

b. Semantic Computation

(i) \[
[[ S ]] =
\]

(ii) \[
[[ \exists e ]] ( [[ vP ]] ) =
\]

(iii) \[
[ \lambda P_{<\text{st}>}. \text{There exists an event e such that } P(e) = T ] ( [[ vP ]] ) =
\]

(iv) There exists an event e such that \[[ vP_1 ]](e) = T =

(v) There exists an event e such that \[[ vP_2 ]](Bill)(e) = T =

(vi) There exists an event e such that \[[ v ]]([[VP]])(Bill)(e) = T =

(vii) There exists and event e such that
\[
[ \lambda P_{<\text{st}>}. [ \lambda x. [ \lambda e . P(e) & Agent(e) = x ] ] ]([[VP]])(Bill)(e) = T =
\]

(viii) There exists and event e such that \[[VP]](e) & Agent(e) = Bill =

(ix) There exists an event e such that
\[
[[V]](Sue)(e) & Agent(e) = Bill =
\]

(x) There exists an event e such that
\[
[ \lambda y. [ \lambda e . \text{kiss}(e,y) ] ] (Sue)(e) & Agent(e) = Bill =
\]

(xi) There exists an event e such that \text{kiss}(e,Sue) & Agent(e) = Bill

That’s all for now!....

Now let’s begin our feature presentation…