An Introduction to Minimalist Grammars

Gregory M. Kobele

Jens Michaelis

Humboldt-Universität zu Berlin
University of Chicago

Universität Bielefeld

ESSLLI 2009
Bordeaux

Semantics

- We briefly turn our attention to the problem of specifying the role the sentences of our grammar play in inference – semantics!
- Warning:
  - The standard way to do semantics in minimalism is to interpret the *derived* structures
  - This is to be contrasted with another natural way of looking at matters, according to which the meaning of an expression is compositionally determined via its derivation

Heim and Kratzer (1998)

- Given a subtree α, with immediate daughters β, γ, the interpretation of α, $[\![\alpha]\!]$, is
  - $[\![\beta]\!]([\![\gamma]\!])$,
    if $[\![\beta]\!] : \sigma \rightarrow \tau$ and $[\![\gamma]\!] : \sigma$
  - $[\![\gamma]\!]([\![\beta]\!])$,
    if $[\![\beta]\!] : \sigma$ and $[\![\gamma]\!] : \sigma \rightarrow \tau$
- Note that in many cases, we can compute the meanings of constituents as we build them up:

$$[\![\textbf{merge}(\beta, \gamma)]\!] = \begin{cases} [\![\beta]\!]([\![\gamma]\!]) \\ \text{or} \\ [\![\gamma]\!]([\![\beta]\!]) \end{cases}$$

Example

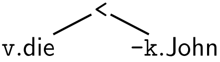
Semantics

- For the time being, we treat **move** as semantically vacuous:

$$[\![\textbf{move}(\alpha)]\!] = [\![\alpha]\!]$$

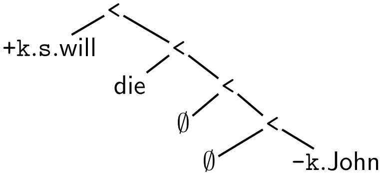
- We see that we can assign types relatively straightforwardly to our lexical items:
  - *seem* : $t \rightarrow t$
  - *die* : $e \rightarrow t$
  - *expect* : $e \rightarrow t \rightarrow t$
  - *kill* : $e \rightarrow e \rightarrow t$
  - *rain* : t
  - *John* : e
  - *-en* : $(e \rightarrow t) \rightarrow t$
- All the other lexical entries let's agree to treat as semantically vacuous (so tense, aspect, etc are being ignored)

Semantics



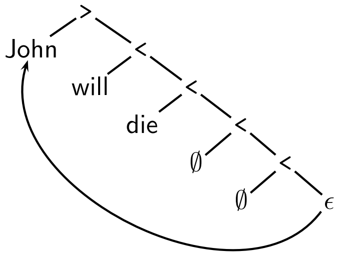
DIE(J)

Semantics



$\text{DIE}(\text{J})$

Semantics



$\text{DIE}(\text{J})$

Semantics

- $[\![\text{John seems to have died}]\!] = \textsc{seem}(\textsc{die}(\textsc{j}))$
- $[\![\text{It seems that John has died}]\!] =$ *same as above*

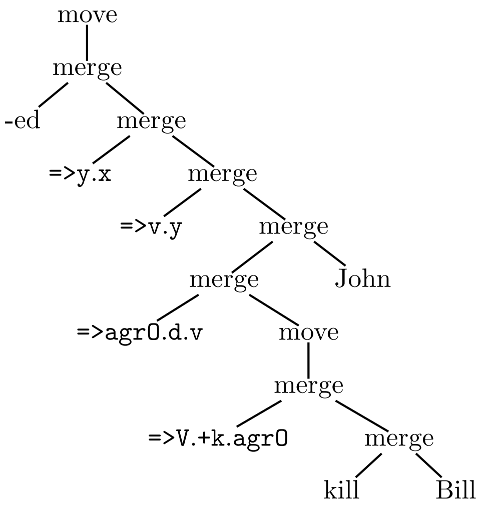
- $[\![\text{John killed Bill}]\!] = \textsc{kill}(\textsc{b})(\textsc{j})$
- $[\![\text{Bill was killed}]\!] = \textsc{en}(\textsc{kill}(\textsc{b}))$

- $[\![\text{Bill seems to have been killed}]\!] = \textsc{seem}(\textsc{en}(\textsc{kill}(\textsc{b})))$

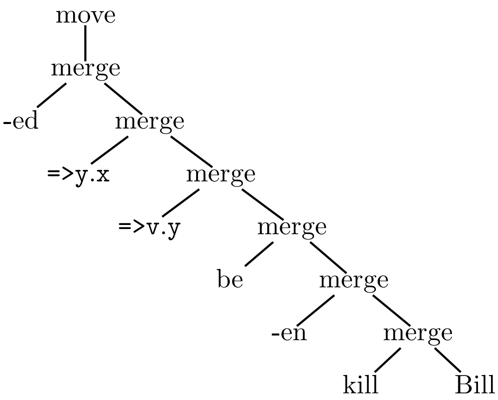
- $[\![\text{John expects Bill to die}]\!] = \textsc{expect}(\textsc{die}(\textsc{b}))(\textsc{j})$
- $[\![\text{Bill is expected to die}]\!] = \textsc{en}(\textsc{expect}(\textsc{die}(\textsc{b})))$

- $[\![\text{Bill is expected to have been killed}]\!] = \textsc{en}(\textsc{expect}(\textsc{en}(\textsc{kill}(\textsc{b}))))$

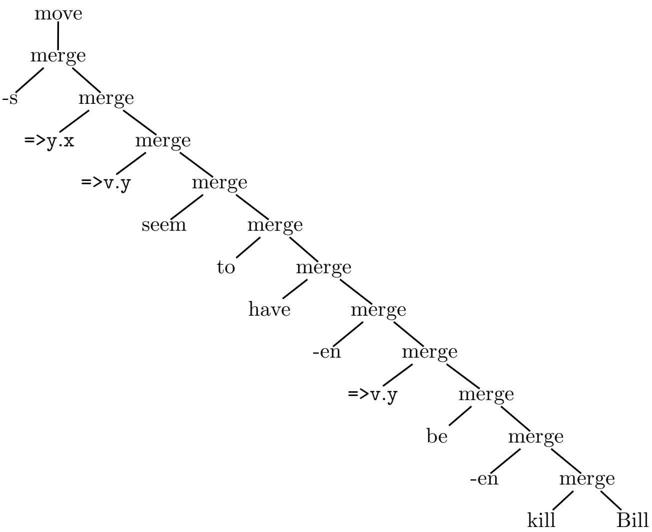
Example


move
merge
-ed
merge
=>y.x
merge
=>v.y
merge
merge
John
=>agrO.d.v
move
merge
=>V.+k.agrO
merge
kill
Bill

Example


move
merge
-ed
merge
=>y.x
merge
=>v.y
merge
be
merge
-en
merge
kill
Bill

Example


move
merge
-s
merge
=>y.x
merge
=>v.y
merge
seem
merge
to
merge
have
merge
-en
merge
=>v.y
merge
be
merge
-en
merge
kill
Bill

Semantics

- The minimalist strategy of inserting expressions into their deep, or semantic, position makes it easy to come up with reasonable predicate argument structures in the semantics
- However, in some cases, the semantic type we want to assign to an expression is not compatible with the semantic type of the expression it first merges with!
  - *everyone* : $(e \rightarrow t) \rightarrow t$
  - *kill* : $e \rightarrow (e \rightarrow t)$
- Although the sentence below is syntactically well-formed, we cannot assign a meaning to it using our current rules!

John killed everyone

Quantifiers

- Remember that our grammar accesses DPs multiple times (twice) during a derivation:
  - once when it is **merge**d into its base position (`d`)
  - and once when it is **move**d into its surface position (`-k`)
- A natural idea is to allow a previously incompatible meaning (such as the quantifier *everyone*) to be attempted to be used whenever it is accessed during a derivation!

$$[\![\text{everyone doesn't seem to have died}]\!] \rightarrow \text{EVERYONE}(\neg(\text{SEEM}(\text{DIE})))$$
$$\rightarrow \neg(\text{SEEM}(\text{EVERYONE}(\text{DIE})))$$

$$[\![\text{It doesn't seem that everyone has died}]\!] \not\rightarrow \text{EVERYONE}(\neg(\text{SEEM}(\text{DIE})))$$
$$\rightarrow \neg(\text{SEEM}(\text{EVERYONE}(\text{DIE})))$$

- How to implement this?

Quantifiers

- We allow moving expressions to optionally be treated for the purposes of **merge** as denoting variables,

$$[\![\textbf{merge}(\beta,\gamma)]\!] = \begin{cases} [\![\beta]\!]([\![\gamma]\!]) \\ \text{or} \\ [\![\gamma]\!]([\![\beta]\!]) \\ \text{or} \\ [\![\beta]\!](x) \quad \text{(if } [\![\gamma]\!] \text{ is a quantifier)} \end{cases}$$

- and then applying their quantificational meaning once they are moved

$$[\![\textbf{move}(\alpha)]\!] = \begin{cases} [\![\alpha]\!] \\ \text{or} \\ [\![\gamma]\!](\lambda x.[\![\alpha]\!]) \qquad \text{(if } \gamma\text{, the moving piece,} \\ \qquad\qquad\qquad\qquad \text{was merged as the variable } x) \end{cases}$$

- This is just a version of cooper storage (Cooper, 1983)!

Quantifiers

- We assume a 'store'; a data-structure containing pairs of variables and functions of type $t \to t$.
- Because we want to keep track of which 'meaning' on the store is associated with which moving constituent, we index the store with features; the SMC guarantees that this relation is functional

$$\textsc{store} : \texttt{Feat} \to Var \times D_{tt}$$

- For S a store and f a feature, we write S/f to denote the store like S but undefined at f
- For S a store, and f, g features, we write $S_{g \leftarrow f}$ to denote the store like S/f but with $S_{g \leftarrow f}(g) = S(f)$
- For S a store, f a feature, and π a pair, $S[f := \pi]$ denotes the store like S but with $S[f := \pi](f) = \pi$
- For S, T stores with disjoint domains, $S \cup T$ is their set theoretic union

Quantifiers

- A minimalist expression will denote a *pair* of objects; $[\![\alpha]\!] = \langle a, A\rangle$ (and so $[\![\beta]\!] = \langle b, B\rangle$, etc). The first component of the pair is its 'normal' meaning, and the second a store

$$[\![\textbf{merge}(\beta,\gamma)]\!] = \begin{cases} \langle b(g),\ B \cup G\rangle \\ \text{or} \\ \langle g(b),\ G \cup B\rangle \\ \text{or} \\ \langle b(x),\ B \cup G[\texttt{-f} := \langle x, g\rangle]\rangle \\ \quad \text{(where } \gamma\text{'s next feature is } \texttt{-f}\text{)} \end{cases}$$

- Given a pair $\langle x, q\rangle$, and a proposition ϕ, $\langle x, q\rangle(\phi)$ is short hand for $q(\lambda x.\phi)$

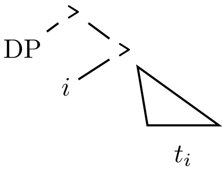
$$[\![\textbf{move}(\alpha)]\!] = \langle A(\texttt{-f})(a),\ A/\texttt{-f}\rangle$$
$$\text{(if } \texttt{-f} \text{ is checked by this operation)}$$

Quantifiers

- $[\![\langle \texttt{V}.\text{die},\ \texttt{-k}.\text{everyone}\rangle]\!] = \textsc{die}(x)$; stored: $\langle x, \textsc{everyone}\rangle$
- $[\![\langle \texttt{+}\underline{\texttt{k}}.\texttt{s}.\text{seems to have died},\ \texttt{-k}.\text{everyone}\rangle]\!] = \textsc{seem}(\textsc{die}(x))$; stored: $\langle x, \textsc{everyone}\rangle$
- $[\![\langle \texttt{s}.\text{everyone seems to have died}\rangle]\!] = \textsc{everyone}(\lambda x.\textsc{seem}(\textsc{die}(x)))$

Quantifiers – Heim and Kratzer (1998)

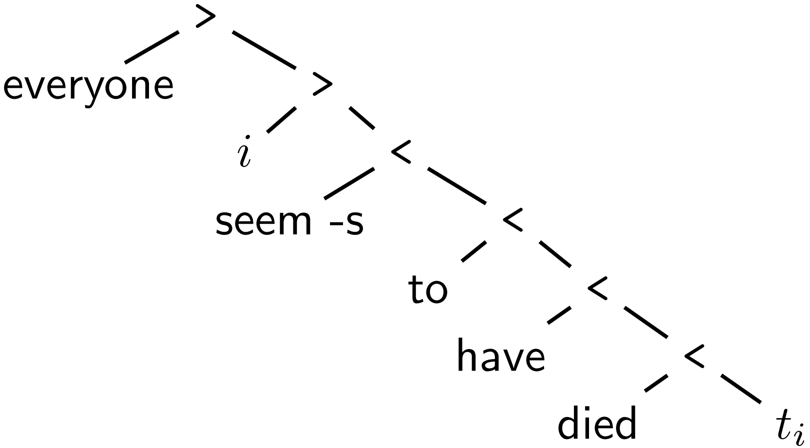
- Although the standard way of interpreting DPs in the generative literature uses (almost) only the standard function application shown before,
- The very same 'cooper storage' idea is present: when a DP is moved, it results in a structure like the below:



- Traces (t_i) are interpreted as variables; $[\![t_i]\!] = x_i$
- Subtrees α of the form $[_> \; i \; \beta]$ are interpreted as follows:

$$[\![[_> \; i \; \beta]]\!] = \lambda x_i.[\![\beta]\!]$$

Example

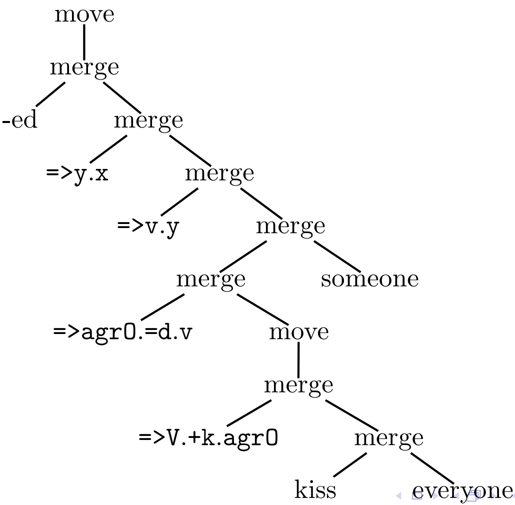


Quantifier Scope

- This account undergenerates:
  1. Everyone doesn't seem to be happy
     - $\checkmark$: $\forall < \neg$
     - $*$: $\neg < \forall$
  2. Someone kissed everyone
     - $*$: $\exists < \forall$
     - $*$: $\forall < \exists$
- For the second reading of example 1, note that the quantifier is merged beneath the negation operator – an idea:
  - The first reading corresponds to interpreting the quantifier in its moved position
  - The second reading corresponds to interpreting the quantifier in its base position
- What about example 2?

Quantifier Scope

- Note that both base and moved-to positions of *everyone* are beneath the base position of *someone*! Thus there is a type-mismatch.



Quantifier Scope

- Our idea is that you can retrieve elements from the store whenever the associated syntactic expression moves
  - Non-surface scope, then, is a consequence of retrieving the meaning of the surface c-commanding element beneath the position where the surface c-commanded element's meaning is retrieved
- But our analysis of sentences like *someone kissed everyone* don't work here!
- Analytical options:
  1. Change theory
  2. Change analysis
- It is not obvious how we should change our theory!
- Our theory tells us how we should change our analysis:
  - The object must move to a position c-commanding the base position of the subject.

Quantifier Scope

- We introduce a new feature type, which is intended to extend the moving domain of objects over the base position of subjects: −q and +q (note: this triggers **agree**/covert movement)
- DPs uniformly have this feature: d.−k.−q
- Where should the licensor variant (+q) go?
  1. Above the base position of the subject (to check the features of the object):

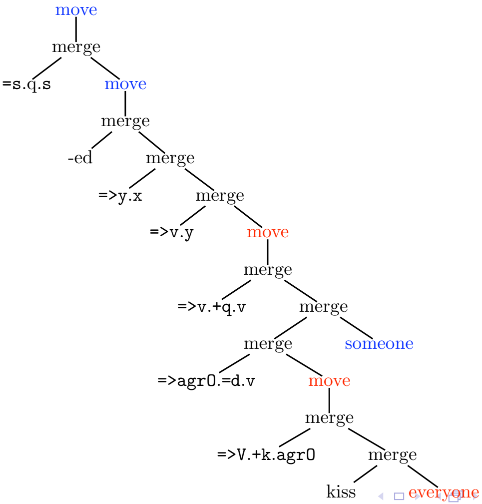
     $$=>v(r).+q.v.\emptyset$$

  2. At the s level (to check the features of the subject):

     $$=s(r).+q.s.\emptyset$$

Quantifier Scope

- Note that the base position of *someone* is beneath the last moved-to position of *everyone*!



Quantifier Scope

- In order for this to work, we have to modify our definition of the interpretation of **move**ment.
- For S a store and f a feature, if S is undefined on f then we write $S(f)$ as shorthand for the identity function.
- $[\![\textbf{move}(\alpha)]\!] \rightarrow$
  1. $\langle A(\texttt{-f})(a),\ A/\texttt{-f}\rangle$
     (if -f is checked by this operation)
  2. $\langle a,\ A_{\texttt{-g}\leftarrow\texttt{-f}}\rangle$
     (if -f is checked by this operation, and the next feature of the moving element is -g)
- We require that:
  - if the moving element is checking its last feature, then the first option apply

Inverse Linking

- One apple in every basket exploded.

  surface: $\exists a.\, \texttt{Apl}(a) \wedge (\forall b.\, \texttt{Bskt}(b) \rightarrow \texttt{In}(a,b)) \wedge \texttt{Expl}(a)$

  inverse: $\forall b.\, \texttt{Bskt}(b) \rightarrow \exists a.\, \texttt{Apl}(a) \wedge \texttt{In}(a,b) \wedge \texttt{Expl}(a)$

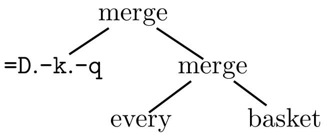
- We will apply our strategy here as well; we will have a -q-driven movement step to a position within the main DP.

Inverse Linking

- Some lexical items for DPs:

=n(r).D.one   =n(r).D.every   n.apple

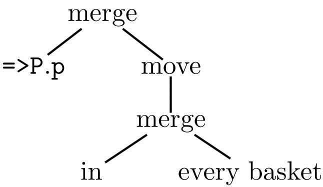
=D(r).d.−k.−q.∅   n.basket



- Some lexical items for PPs:

=d(r).+<u>k</u>.P.in =>P(r).p.∅

=>P(r).+q.P.∅

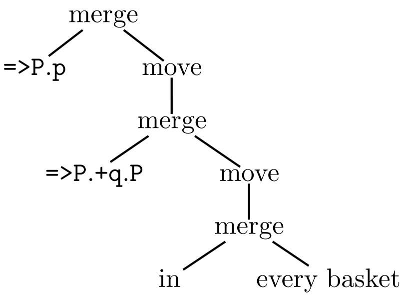


Inverse Linking

- Some lexical items for PPs:

=d(r).+$\underline{\text{k}}$.P.in   =>P(r).p.$\emptyset$

=>P(r).+q.P.$\emptyset$

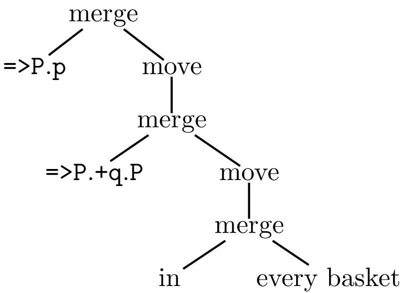


Inverse Linking

- We assign semantic types to these expressions as follows:
  - *every* : $(e \to t) \to t$
  - *some* : $(e \to t) \to t$
  - *basket* : $e \to t$
  - *apple* : $e \to t$
  - *in* : $e \to e \to t$
- All the other (phonetically empty) lexical entries let's agree to treat as semantically vacuous

Inverse Linking

- Our types don't work in the case of the second PP derivation!

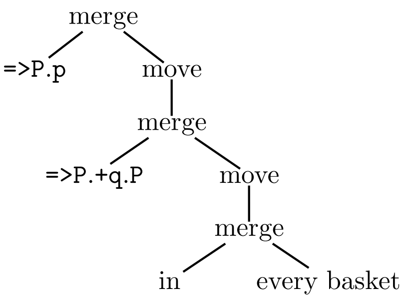


Inverse Linking

- We modify one last time our definition of the interpretation of **move**:
- For S a store and f a feature, if S is undefined on f then we write $S(f)$ as shorthand for the identity function.
- For $\langle x, q\rangle$ an element of a store, and ϕ of type t or of type $\xi \rightarrow t$, where ξ is any type:
  - $\langle x, q\rangle(\phi_t) = q(\lambda x.\phi)$
  - $\langle x, q\rangle(\phi_{\xi t})(a) = q(\lambda x.(\phi(a)))$
- $[\![\mathbf{move}(\alpha)]\!] \rightarrow$
  1. $\langle A(\texttt{-f})(a),\ A/\texttt{-f}\rangle$
     (if -f is checked by this operation)
  2. $\langle a,\ A_{\texttt{-g}\leftarrow\texttt{-f}}\rangle$
     (if -f is checked by this operation, and the next feature of the moving element is -g)
- We require that:
  - if the moving element is checking its last feature, then the first option apply

Inverse Linking

- Now things work:

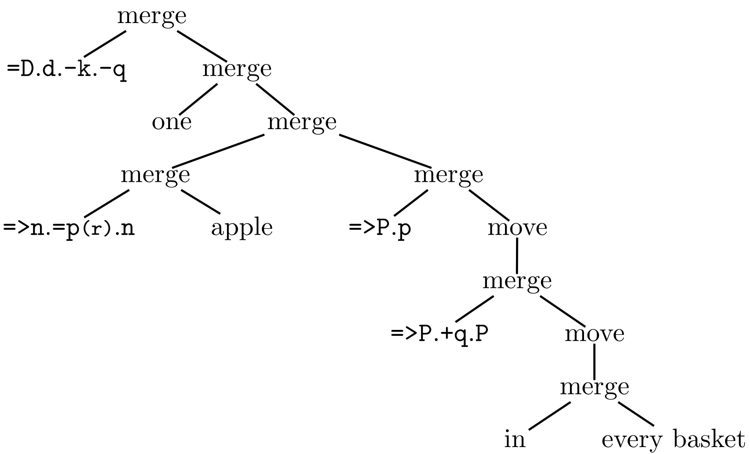


Inverse Linking

- We need lexical items which allow us to put PPs and DPs together:

  =>n(r).=p(r)(r).n.∅

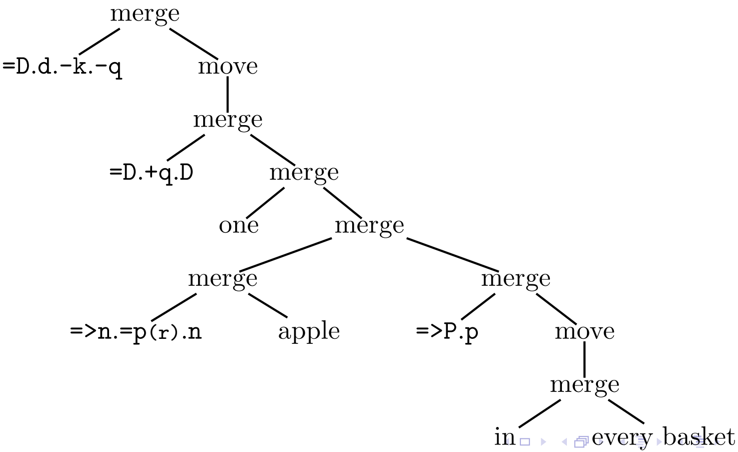
- This item denotes predicate conjunction! $((A\&B)(a) = A(a) \wedge B(a))$



Inverse Linking

- We also allow the DP in the PP to check its −q feature at the D-level, to take scope over its containing DP:

=D(r).+q.D



Cooper, R. (1983). *Quantification and Syntactic Theory*. Dordrecht: D. Reidel.

Heim, I. and A. Kratzer (1998). *Semantics in Generative Grammar*. Blackwell Publishers.