Adding semantics to the grammar

• Logical form
  For each sentence admitted by the grammar, we want to produce a meaning representation suitable for applying rules of inference.
  This fierce dog chased that angry cat.
  \( \text{this}(x) \land \text{fierce}(x) \land \text{dog}(x) \land \text{chased}(e, x, y) \land \text{that}(y) \land \text{angry}(y) \land \text{cat}(y) \)

• Compositionality
  The meaning of a phrase is composed of the meanings of its parts.

• Existing machinery
  Unification is the only mechanism for constructing semantics in the grammar.
  (No relational constraints)
Semantics in feature structures

• Semantic content in the SEM attribute of every word and phrase

\[
\text{syn-struct} = \begin{cases} 
\text{HEAD} & \text{pos} \\
\text{SPR} & \text{*list*} \\
\text{COMPS} & \text{*list*} \\
\text{SEM} & \text{cont}[\text{RELS} *\text{dlist*}] 
\end{cases}
\]

• The value of SEM for a sentence is simply a list of relations in the attribute RELS, with the arguments in those relations appropriately linked:

\[
[\text{RELS} <! [ \text{PRED "the_q"} , [ \text{PRED "dog_n"} , [ \text{PRED "bark_v"}
\text{ARG0 x1 }] \text{ARG0 x1 }] \text{ARG0 e1}
\text{ARG1 x1 }] ] !> ]
\]

• Semantic relations are introduced by lexical entries, and are appended when words are combined with other words or phrases to form larger phrases.
A simple example

```
S
[RELS <! [ PRED "the_q" ], [ PRED "dog_n" ], [ PRED "bark_v"
    ARG0 x1 ] ARG0 x1 ] ARG0 e1
    ARG1 x1 ] !> ]
```

```
NP
[RELS <! [ PRED "the_q" ], [ PRED "dog_n" ]]
[RELS <! [ PRED "bark_v"
    ARG0 x1 ] ARG0 x1 ] !> ]
```

```
Det
[RELS <! [ PRED "the_q"
    ARG0 x1 ] !> ]
```

```
N
[RELS <! [ PRED "dog_n"
    ARG0 x1 ] !> ]
```

```
ARGO e1
ARG1 x1 ] !>]
```

```
VP
[RELS <! [ PRED "bark_v"
    ARG0 e1
    ARG1 x1 ] !>]
```
Appending lists with unification

• A *difference list* embeds an open-ended list into a container structure that provides a ‘pointer’ to the end of the ordinary list.

\[
\begin{align*}
\text{A} & \quad \text{LIST} \, 1 \quad \text{FIRST \ "item-1"} \\
& \quad \text{REST} \, 2 \quad \text{\ast \ list\!\!\!*} \\
& \quad \ast\text{dlist}\ast \quad \text{LAST} \, 2 \\
\text{B} & \quad \text{LIST} \, 3 \quad \text{FIRST \ "item-2"} \\
& \quad \text{REST} \, 4 \quad \text{\ast \ list\!\!\!*} \\
& \quad \ast\text{dlist}\ast \quad \text{LAST} \, 4
\end{align*}
\]

• Using the LAST pointer of difference list A we can append A and B by
  (i) unifying the front of B (i.e. the value of its LIST feature) into the tail of A (its LAST value) and
  (ii) using the tail of difference list B as the new tail for the result of the concatenation.
Appending lists

\[
\begin{align*}
\text{LIST} & \quad \text{REST} \\
1 & \quad *\text{ne-list}* \\
\text{FIRST} & \quad \text{REST} \\
"item-1" & \quad 2 \quad \text{*list*} \\
\text{LAST} & \quad *\text{ne-list}* \\
2 & \quad \text{FIRST} \\
\end{align*}
\]
Appending lists

\[
\begin{align*}
A &\quad \text{LIST } 1 \\
&\quad \text{LAST } 2 \\
&\quad \text{*dlist*}
\end{align*}
\]

\[
\begin{align*}
&\quad \text{FIRST } "\text{item-1"} \\
&\quad \text{REST} \\
&\quad \text{*ne-list*} \\
&\quad \text{*ne-list*} \\
\end{align*}
\]

\[
\begin{align*}
&\quad \text{FIRST } "\text{item-2"} \\
&\quad \text{REST} \\
&\quad \text{2 *list*} \\
\end{align*}
\]

\[
\begin{align*}
B &\quad \text{LIST } 3 \\
&\quad \text{LAST } 4 \\
&\quad \text{*dlist*}
\end{align*}
\]

\[
\begin{align*}
&\quad \text{FIRST } "\text{item-3"} \\
&\quad \text{REST} \\
&\quad \text{4 *list*} \\
\end{align*}
\]
Linking semantic arguments

- Each word and phrase also has an INDEX attribute in SEM
- When heads select a complement or specifier, they also constrain its INDEX value – an instance variable for noun, or an event variable for verbs.
- Each lexeme also specifies a KEY relation (to allow for complex lexical semantics)
Semantics of phrases

- Every phrase makes the value of its own `RELS` attribute be the result of appending the `RELS` lists of its daughter(s), using unification of difference lists.

- Every phrase identifies its semantic `INDEX` value with the `INDEX` value of one of its daughters (the semantic head).

- Since we unify the whole `syn-struc` of a complement or specifier with the constraints in the head-daughter, unification also takes care of semantic linking.

- Head-modifier structures work analogously – the modifier lexically constrains the semantic index of the head-daughter it will modify, and then the rules unify the whole `syn-struc` of the head-daughter with the `MOD`’s value in the modifier.
Our simple example again

S

[INDEX e1
  RELS <! [ PRED "the_q" , [ PRED "dog_n" , [ PRED "bark_v"
      ARGO x1 ] ARG0 x1 ] ARGO e1
      ARG1 x1 ] !> ]

  /
  /
  /

NP VP

[INDEX x1
  RELS <! [ PRED "the_q" , [ PRED "dog_n" RELS <! [ PRED "bark_v"
      ARGO x1 ] ARG0 x1 ] !> ]

  /
  /
  /

Det N

[INDEX x1
  RELS <! [ PRED "the_q"
      ARGO x1 ] !> ]

  /
  /
  /

[INDEX x1
  RELS <! [ PRED "dog_n"
      ARGO x1 ] !> ]