

Topics in Computational Linguistics — Grammar Engineering —

Dan Flickinger

CSLI Stanford & Saarland University

`danf@csli.stanford.edu`

Stephan Oepen

Universitetet i Oslo & CSLI Stanford

`oe@csli.stanford.edu`

<http://lingo.stanford.edu/courses/05/ge/>

Orthographic Variation: 'Inflectional' Rules

```
%(letter-set (!s abcdefghijklmnopqrstuvwxyz))
```

```
noun-non-3sing_irule :=
```

```
%suffix (!s !ss) (!ss !ssses) (ss sses)
```

```
word &
```

```
[ HEAD [ AGR non-3sing ],
```

```
  ARGS < noun-lxm > ].
```

```
noun-3sing_irule :=
```

```
word &
```

```
[ ORTH #1,
```

```
  HEAD [ AGR 3sing ],
```

```
  ARGS < noun-lxm & [ ORTH #1 ] > ].
```

dog

|

dogs

bus

|

busses

pass

|

passes



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.

$this(x) \wedge fierce(x) \wedge dog(x) \wedge chased(e,x,y) \wedge that(y) \wedge angry(y) \wedge cat(y)$

- **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

expression $\left[\begin{array}{l} \text{HEAD } pos \\ \text{SPR } *list* \\ \text{COMPS } *list* \\ \text{SEM } cont \left[\text{RELS } *dlist* \right] \end{array} \right]$

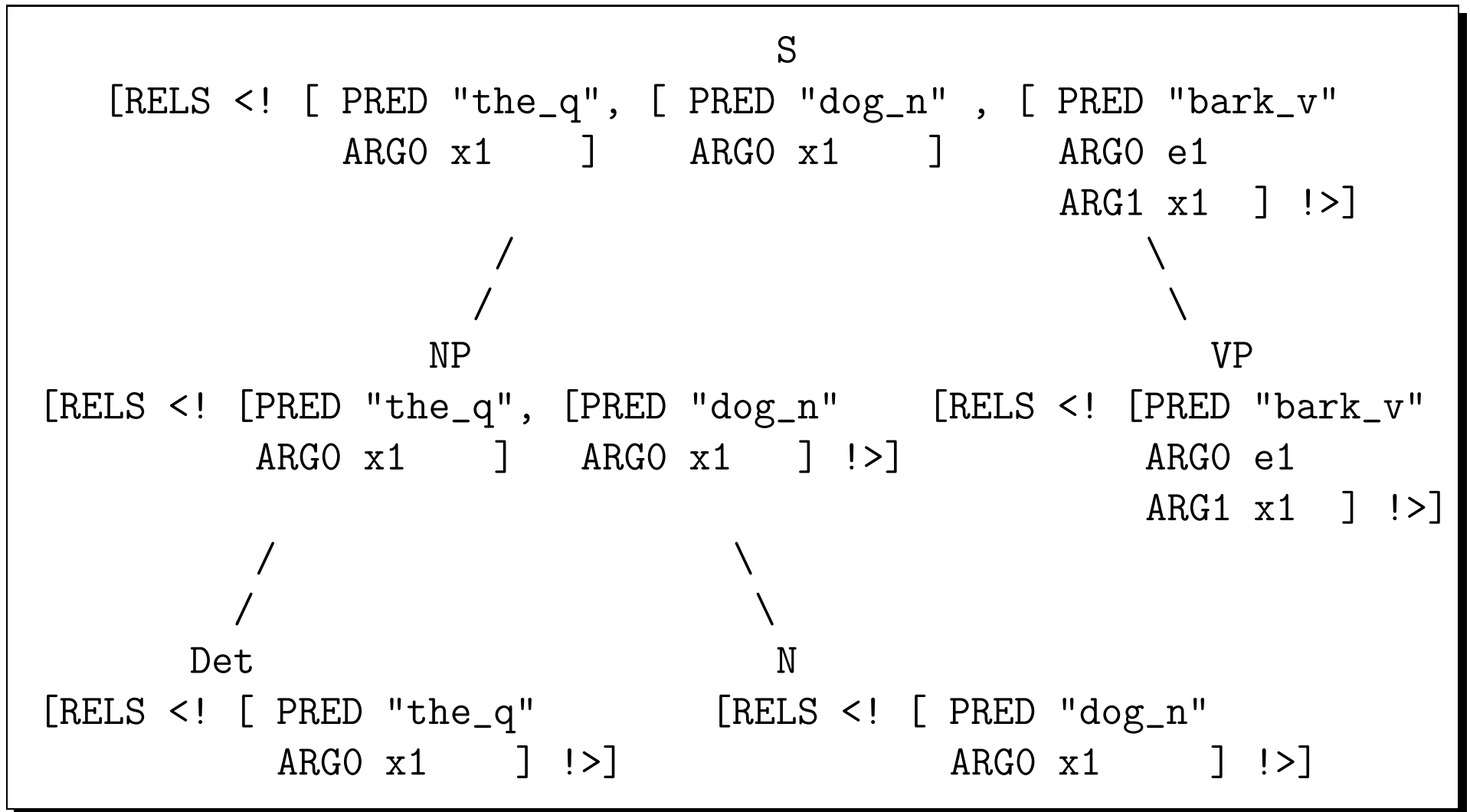
- The value of SEM for a sentence is simply a list of relations in the attribute RELS, with the arguments in those relations appropriately linked:

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

- Semantic relations are introduced by lexical entries, and are appended when words are combined with other words or phrases.

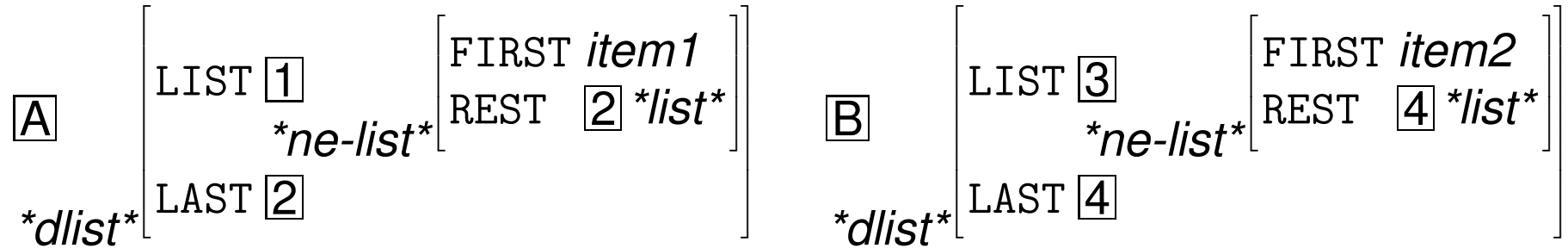


A simple example



Appending lists with unification

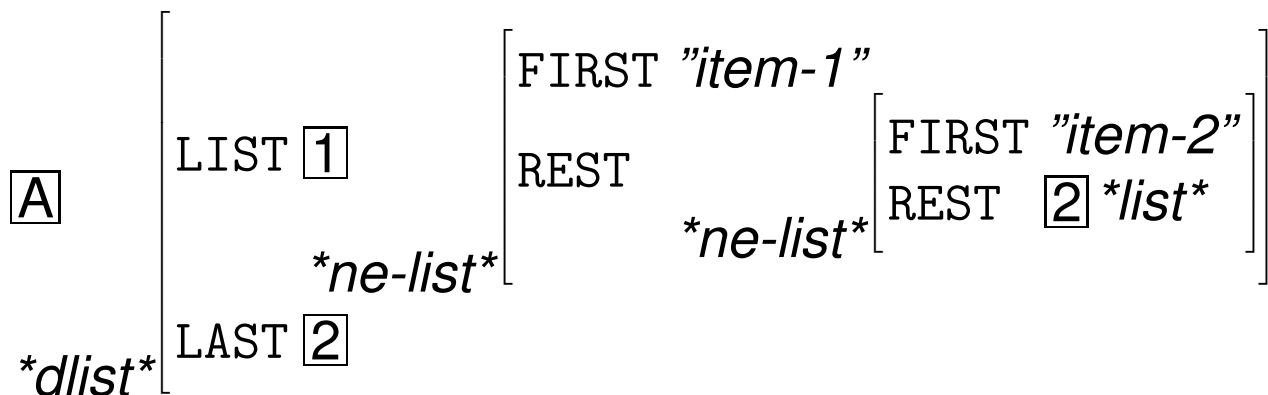
- A *difference list* embeds an open-ended list into a container structure that provides a ‘pointer’ to the end of the ordinary list.



- 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.



Result of appending lists



Linking semantic arguments

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

	HEAD	<i>verb</i>	
	SPR.FIRST	[SEM. INDEX 1]	
	COMPS.FIRST	[SEM. INDEX 2]	
		INDEX 0 <i>event</i>	
		KEY	[PRED <i>string</i>
			ARG0 0
			ARG1 1
			ARG2 2
		3 <i>arg12-v-rel</i>	
		RELS <! 3 !>	
<i>trans-verb-lxm</i>	SEM		



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-struct* 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-struct* of the head-daughter with the `MOD`'s value in the modifier.



Our simple example again

