Finnish in Trale

Seminar: "Grammar Engineering"

Kristiina Muhonen

muhonen@sfs.uni-tuebingen.de

University of Tübingen, Seminar für Sprachwissenschaft

July 3, 2006

Overview

1. *Facts about Finnish:*
   - Null-subject Sentences
   - Subjectless Sentences
   - Passive
   - Word Order and Discourse Patterns

2. *Basic Grammar for Finnish:*
   Kasprzik (2005): “Implementierung einer kleinen HPSG- Grammatik für das Finnische in TRALE”

Null-subject Sentences

Finnish is a partial pro-drop language:

- 1st and 2nd person pronouns in subject position are optional
- Subject information can be extracted from the predicative because of sufficient agreement

  (Minä) menen kauppaan.
  I-NOM go-1.P.Sg store-ILL

  'I go to the store.'

- Subject is phonologically empty

Subjectless Sentences

- No visible generic pronoun (man, one...)
- If the subject is unknown, 2 constructions are possible:

1. *Generic sentences:* Arbitrary subject

   Sieltä saa kahvia.
   from there get-3.P.Sg. coffee

   'One gets/can get coffee from there.'

2. *Passive:* Speaker usually ruled out

   Kahvia haetaan sieltä.
   coffee-PART fetch-PASS from there

   'One fetches coffee from there.'

Passive

- A prominent example of a completely subjectless sentence

  - Patient does not act as the subject
  - Agent is not represented at all –> undetermined

- Takes only one form
- . . . could not agree with a possible subject

Passive with the direct object

- Active object ACC –> Passive object ACC2
- Nouns and impersonal pronoun, "it", ACC2=nom
- Pronouns, ACC2=ACC

Otan sen	Se otetaan
take-1.P.Sg. it-ACC	it-NOM take-PASS
’I take it.’	’It is taken.’
Otan sinut.	Sinut otetaan.
take-1.P.Sg. you-ACC	you-AKK take-PASS
’I take you.’	’You are taken.’

Word Order

- Information expressed
- . . . English by syntactic structure
- . . . Finnish by morphological affixation
  - POS and syntactic function embedded in the word
  - Word order can be changed without changing the core meaning of the sentence
- Different syntactic orders reflect a discourse counfigurational structure
- Constituents in certain positions convey a specific discourse function

Discourse Patterns

	Contrast	*Topic*	*Verb*	*Rest*	*English Equivalent*
1.		Leena	söi	omenan.	Leena ate the apple.
2.		Omenan	söi	Leena.	The apple was eaten by Leena.
3.	Omenan	Leena	söi.		It was the apple that Leena ate.
4.	Leena	omenan	söi.		It was Leena who ate the apple.
5.	Söi	Leena		omenan.	Leena DID eat the apple.
6.	Söi	omenan		Leena.	??awkward??

- 1= canonical word order, subject=Topic, object=Rest
- 3&4 correct the previous discourse. Subject/Object=C, contrasting a Subject/Object mentioned before
- 5&6 special argumentative character, verb=C, insisting on the truth

Basic Grammar for Finnish (1)

"Studienarbeit" by Anna Kasprzik

- Following the English Core Fragment of the Grammar Formalisms and Parsing Book [section 3.2.]
- Modified greatly to match the needs of Finnish
- . . . details added, *e.g. Cases*
- . . . details removed, *e.g. Unlocal Dependencies*
- Several versions to cover different phenomena

Basic Grammar for Finnish (2)

Different versions of the grammar expand the coverage:

1. Core Fragment (partly from section 3.2., G&P):
   - Subjectless sentences
   - Null-subject sentences
2. Core Fragment expanded to cover passivization
3. Core Fragment + Passivization + Word order variation
   - Two different approaches:
     - Several phrase structure rules
     - One phrase structure rule with predicates

The Signature

- Some parts could be directly taken over from the English signature [section 3.2.1.1]
- Some attributes and types removed
- Some added, e.g. `pron`, new semantic relations, subtype for `ref`
- Append\3 Relation
- Lexical elements

Basic fragment

Dealing with the optional 1st and 2nd person pronouns to cover null-subject sentences:

```
minä ~~> (synsem:(category:head:(pron, case:nom),
                  content:index:(realref, person:first,
                                 number:sing))).


empty  word,
       phon:e_list,
       synsem:(category:head:(pron, case:nom),
               content:index:(realref, person:first,
                              number:sing)).
```

Two lexical entries for:

- 1st and 2nd person pronouns
- irreferential 3rd person “it”

Basic fragment (2)

Dealing with the subjectless (generic and passive) sentences

From the signature:

```
index person:person number:number gender:gender
   ref
     realref
     man
   it
```

The phonologically empty “one” in generic and passive sentences has

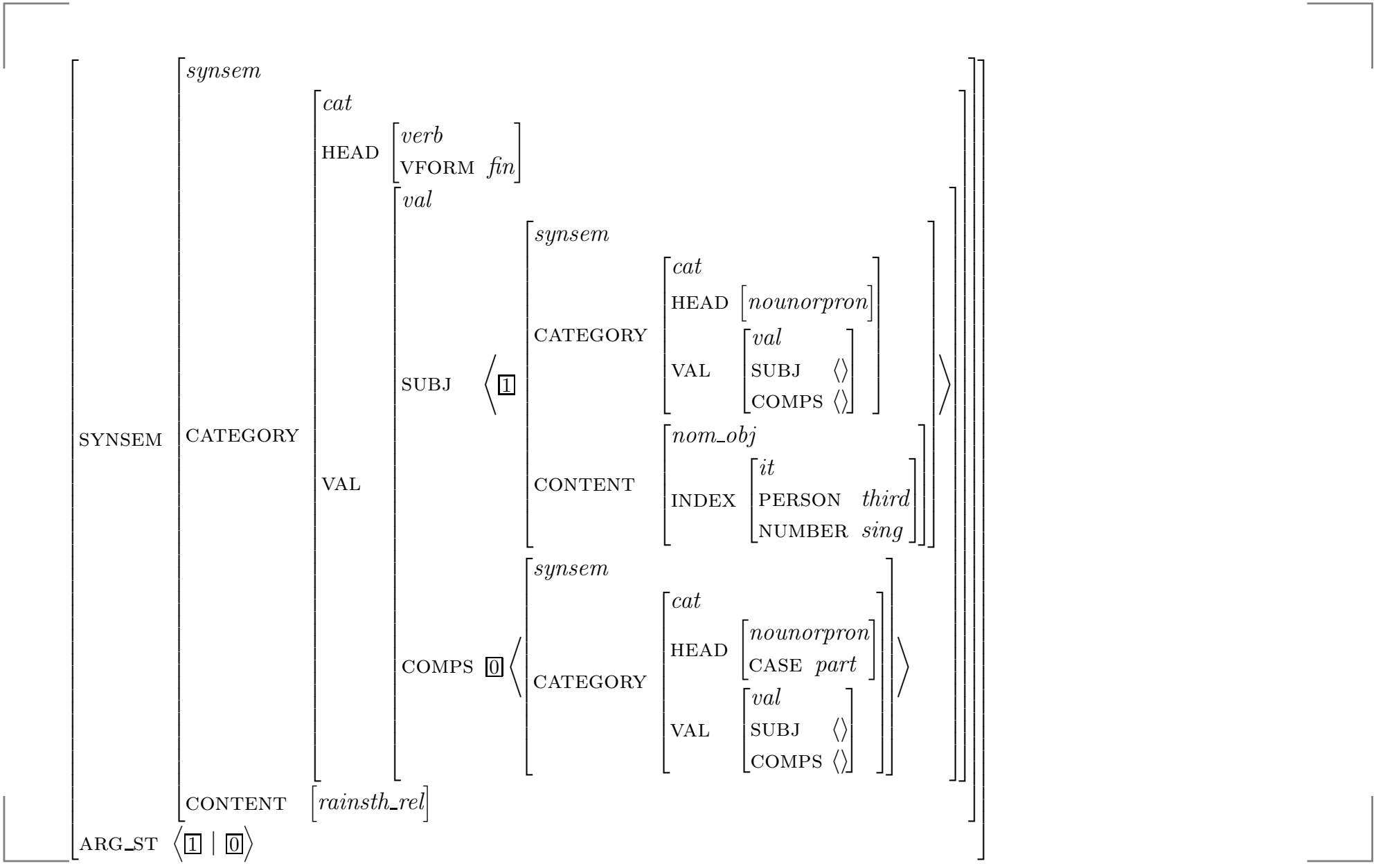
```
empty  word,
       phon:e_list,
       synsem:(category:head:(pron, case:nom),
               content:index:(man, person:third,
                                   number:sing)).
```

Basic fragment (3)

- The signature encodes the nonreferential pronoun “it”
- Used with “sataa”, *it rains*

```
empty  word,
       phon:e_list,
       synsem:(category:head:(pron, case:nom),
               content:index:(it, person:third,
                                  number:sing)).



(word, synsem:content:rainsth_rel) *>
(arg_st:[(content:index:it),(category:head:case:part)]).

(word, synsem:content:rain_rel) *>
(arg_st:[(content:index:it)]).
```

AVM for “sataa”, *it rains*



Basic Principles

On the basis of [section 3.2.1.1, G&P]

- *Structural Case Principle* (only for the subject)

```
Synsemˆ(phrase,
 daughters:(hdtr:synsem:category:(head:vform:fin,
                                  val:subj:[Synsem]),
            ndtr:synsem:(Synsem,
                         category:head:noun)))
*> (daughters:ndtr:synsem:category:head:case:nom).
```

- The infix operator, ˆ, is almost like an existential quantifier in TRALE
  - . . . ensures that `Synsem` is always the same variable

Basic Principles (2)

- *Argument Realization Principle*

```
% - ARP I
(word, synsem:category:head:(verb, vform:(fin;inf))) *>
(synsem:category:val:(subj:[Synsem], comps:List), arg_st:[Synsem|List]).

% - ARP II for the subjectless passive
(word, synsem:category:head:(nounorpron; (verb, vform:pas))) *>
(synsem:category:val:(subj:e_list, comps:List), arg_st:List).
```

- Simpler than the original version
- `subj` and `comps` already specified in LEs of verbs
- ARP has to only take care of `arg_st`

Basic Grammar + Passivization

Additions to the signature

- new verb forms:

```
vform
   fin
   pas
   inf  --> person and number ‘‘neutral’’ verb form
```

- new cases:

```
   case
      nom
      acc
      acc2
      part
      all
      abl
```

Basic Grammar + Passivization (2)

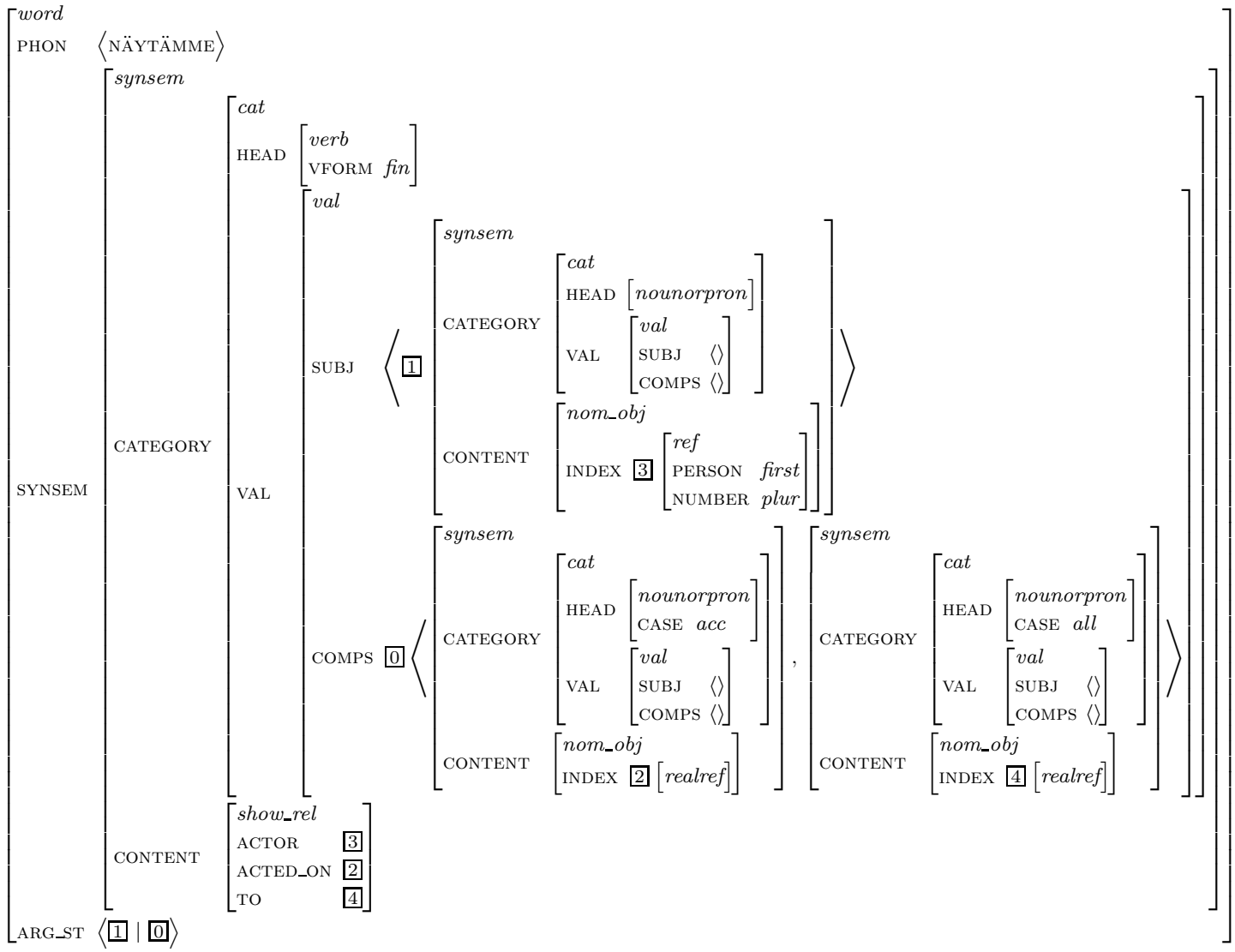
- Changes to the semantic relations, (`cont:psoa`):

```
walk_rel walker:ref
rain_rel
bin_rel
  love_rel lover:ref loved:realref
  eat_rel eater:ref eaten:realref
  get_rel getter:ref gotten:realref
  rainsth_rel rained:realref
```

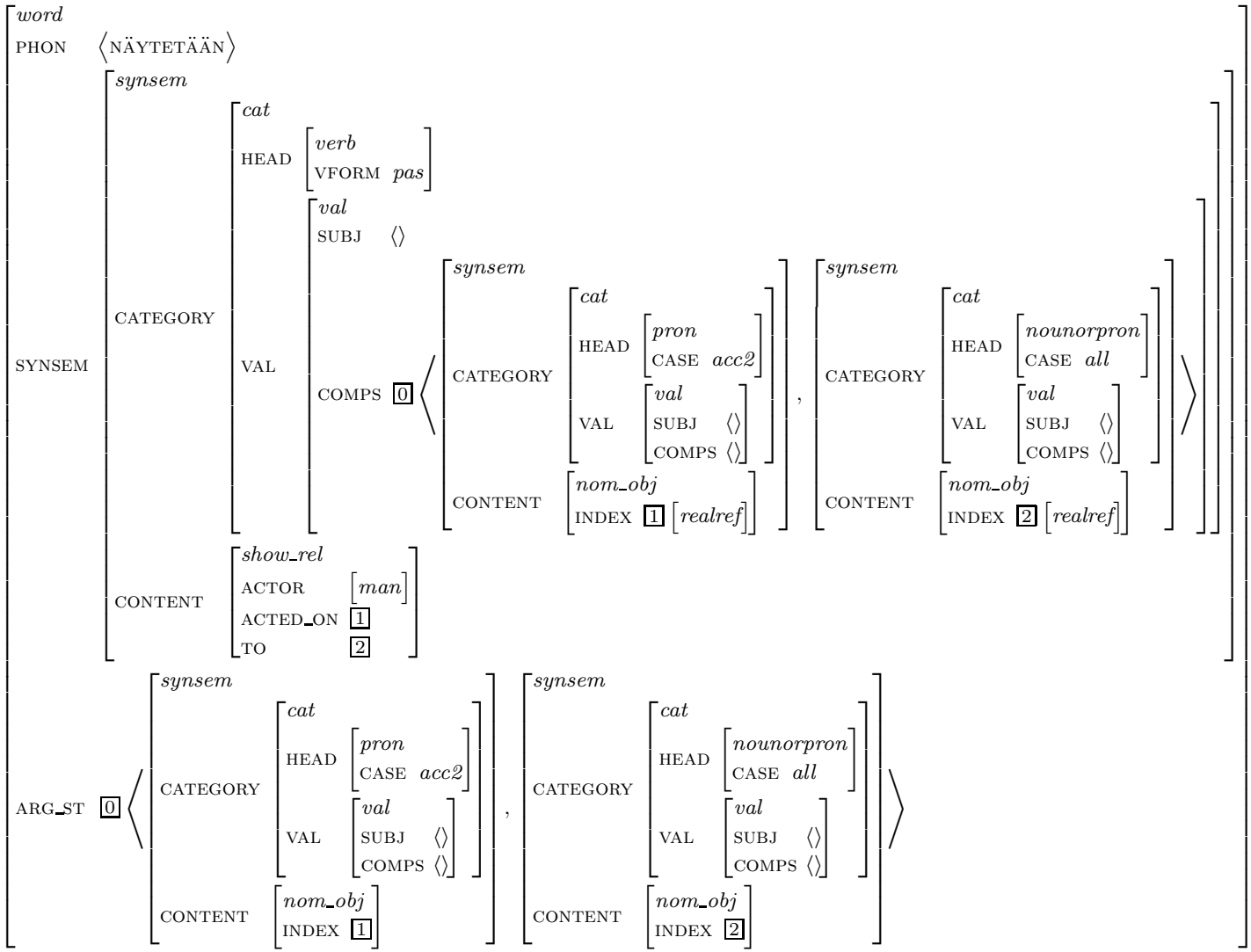
–> Add
- `actor`,
- `acted_on`,
- `to/from`
- ditransitive verbs

```
nonact_rel
   rain_rel
   rainsth_rel material:realref
action_rel actor:ref
   un_rel
      walk_rel
       trans_rel acted_on:realref
          love_rel
          eat_rel
          get_rel
          tri_rel
            to_rel to:realref
               give_rel
               show_rel
            from_rel from:realref
               take_rel
```

Passivization example



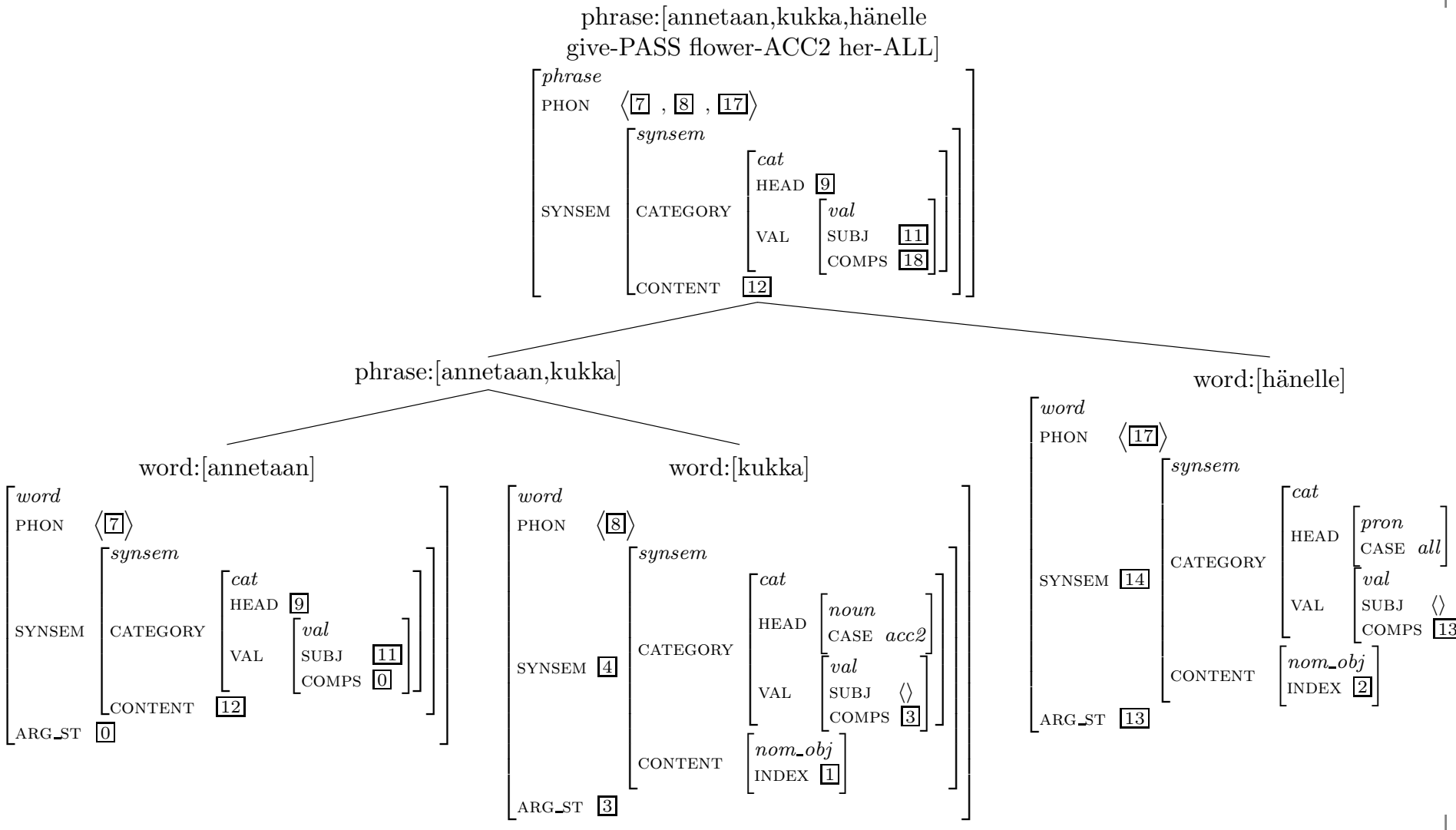
Passivization example



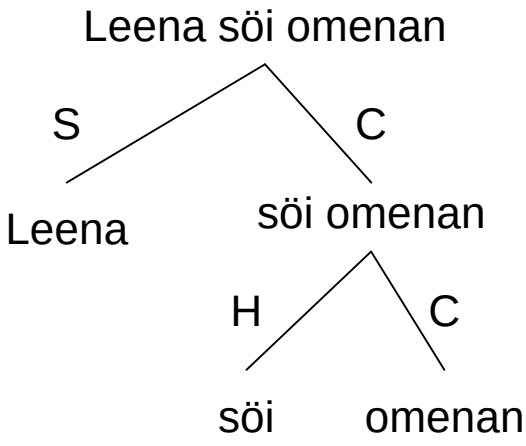
Passive lexical rule

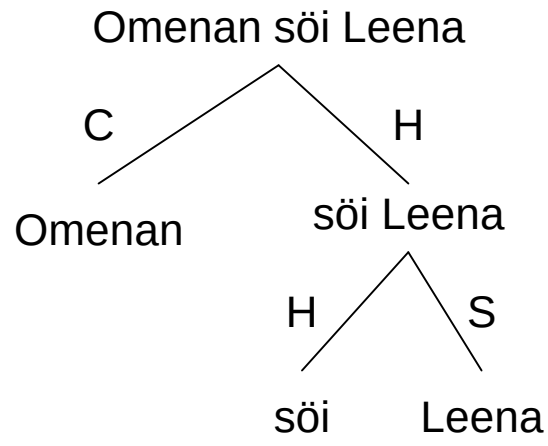
```
passive_lex_rule ##
(word, synsem:(category:(head:(verb, vform:inf),
 val:(subj:[_], comps:L1)),
       content:(action_rel, Y))) **>
(word, synsem:(category:(head:(verb, vform:pas),
 val:(subj:e_list, comps:L2)),
       content:(Y, actor:man)))
if argswap(L1,L2)

morphs
kävellä becomes kävellään, etc

argswap(e_list,e_list) if true.
argswap([(category:head:(H1, case:C), content:E)|List],
        [(category:head:(H2, case:D), content:E)|List]) if
        (casetrafo(C,D), ((H1=noun, H2=noun);(H1=pron, H2=pron))).

casetrafo(part,part) if true.
casetrafo(acc,acc2) if true.
```

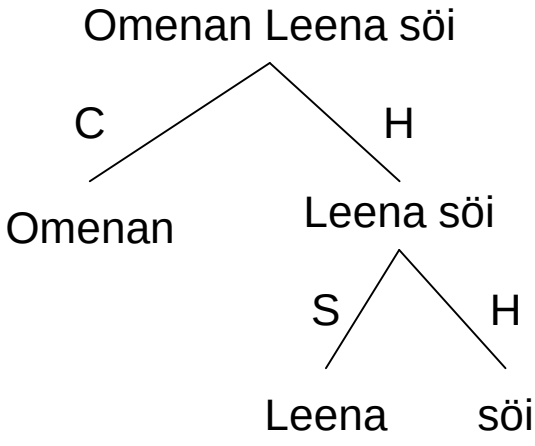
Passivization example

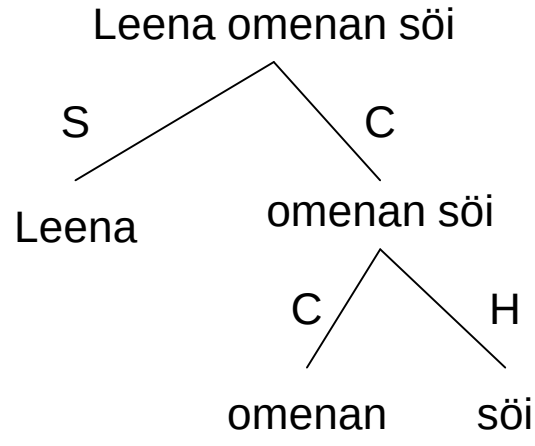


Free Word Order: Motivation





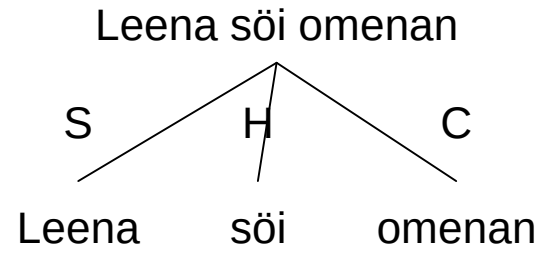




Implementing Word Order

First and biggest step:

Moving from binary to flat structure:



Modifying the signature

- `hs_struc` & `hc_struc` –> `harg_struc`
- `ndtr` –> - `ndtrs` (list of non-daughters)
- `fdtr` (front daughter, can be empty)

```
Old signature:  const_struc hdtr:sign ndtr:sign
                            hs_struc
                            hc_struc

New signature:  const_struc fdtr:list hdtr:sign ndtrs:list
                             harg_struc
```

Implementing Word Order

Two approaches:

1. several phrase structure rules
   - a separate rule for every permutation

2. one phrase structure rule
   - predicates to handle all the permutations

Schemas for Word Order

Head-Argument-Schema:

- replaces *Head-Subject-* and *Head-Complement-Schemas*
- *distribute* checks that the `synsem` values of all elements on the list are connected.
  Eventually permutates the same list.

$$\begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{CATEGORY} & \begin{bmatrix} \text{VAL} & \begin{bmatrix} \text{SUBJ} & \langle\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{DAUGHTERS} & \begin{bmatrix} \textit{harg-struc} & \\ \text{FDTR} & \boxed{3} \\ \text{HDTR} & \begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{CATEGORY} & \begin{bmatrix} \text{VAL} & \begin{bmatrix} \text{SUBJ} & \boxed{1} \\ \text{COMPS} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{NDTRS} & \boxed{4} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\wedge\ \textit{distribute}\ (\ \boxed{1}, \boxed{2}, \boxed{3}, \boxed{4})$$

Principles for Word Order

Semantics Principle

```
OLD:
phrase *> (synsem:content:Content,
           daughters:((hs_struc;hc_struc), hdtr:synsem:content:Content)).
NEW:
phrase *> (synsem:content:Content,
           daughters:(harg_struc, hdtr:synsem:content:Content)).
```

Principles for Word Order (2)

Constituent Order Principle

- takes care of the new varied word order
- *extractappend* checks that all the phon values of the elements on the second list match with them of the first

$$\left[\textit{phrase} \right] \Longrightarrow \left[\begin{array}{ll} \textsc{phon} & \boxed{1} \oplus \left\langle \boxed{2} \right\rangle \boxed{3} \\ \textsc{daughters} & \left[\begin{array}{ll} \textsc{fdtr} & \boxed{4} \\ \textsc{hdtr} & \left[\textsc{phon} \ \boxed{2} \right] \\ \textsc{ndtrs} & \boxed{5} \end{array} \right] \end{array} \right]$$
$$\wedge \ \textit{extractappend} \ \left(\boxed{1}, \boxed{4} \right) \wedge \textit{extractappend} \left(\boxed{3}, \boxed{5} \right)$$

Principles for Word Order (3)

Verb initial (5&6) active sentences in Finnish rather special

	Contrast	*Topic*	*Verb*	*Rest*	*English Equivalent*
1.		Leena	söi	omenan.	Leena ate the apple.
2.		Omenan	söi	Leena.	The apple was eaten by Leena.
3.	Omenan	Leena	söi.		It was the apple that Leena ate.
4.	Leena	omenan	söi.		It was Leena who ate the apple.
5.	Söi	Leena		omenan.	Leena DID eat the apple.
6.	Söi	omenan		Leena.	??awkward??

Active Front Filling Principle makes sure that active sentences do not start with a verb.

The principle is actually more of a style matter.

Active Front Filling Principle

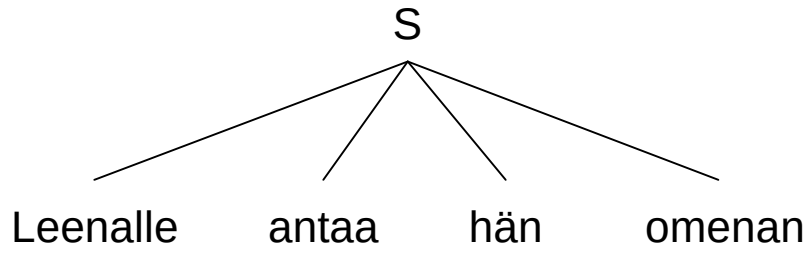
```
(phrase, synsem:category:head:vform:fin) *>
          (daughters:fdtr:ne_list).

fun list_of_signs(-)
list_of_signs(X) if when ((X=e_list;ne_list)),
                  und_list_of_signs(X)).
und_list_of_signs(X) if (X=e_list).
und_list_of_signs(X) if (X=[sign|Y]), list_of_signs(Y).

%ndtrs:list(sign)
%fdtr:list(sign), which can be empty or have one element.
const_struc *> (fdtr:list_of_signs, ndtrs:fdtr list_of_signs).
const_struc *> fdtr:([];[_])
```

Flat Structure

With the *The Active Front Filling Principle* we get structures like this.
The structure is completely flat:

Leenalle	antaa	hän	omenan.
Leena-ALL	give-3.P.Sg.	s/he-NOM	apple-ACC

’S/he gives an apple to Leena (... and a carrot to me).’



Phrase Structure Rules for Word Order

To achieve the desired flexibility in the word order, there are several phrase structure rules that take care of permutating every order.

A few examples:

1.

```
% Ex: ‘‘Leena kävelee‘‘ - Leena-NOM walk-3.P.Sg.
head_one_rule ##
(phrase, synsem:category:val(subj:e_list, comps:e_list),
 daughters:(harg_struc, fdtr:Fdtr,
                  hdtr:(Hdtr, synsem:category:val:(subj:[Synsem],
                                                   comps:e_list)),
                  ndtrs:Ndtrs))  ===>
 cats> (Fdtr, [(synsem:Synsem)]),
 cat> Hdtr,
 cats> (Ndtrs, e_list).
```

Phrase Structure Rules for Word Order

2.

```
% Ex: ‘‘Leenalle annamme me omenan ’’- Leena-ALL give-1.P.PL we-NOM an app
head_3indsdir_rule ##
(phrase, synsem:category:val(subj:e_list, comps:e_list),
 daughters:(harg_struc, fdtr:Fdtr,
                  hdtr:(Hdtr, synsem:category:val:
                          (subj:[Synsem], comps:[CSynsem1,CSynsem2])),
                  ndtrs:Ndtrs))  ===>
 cats> (Fdtr, [(synsem:CSynsem2)])
 cat> Hdtr,
 cats> (Ndtrs, [(synsem:Synsem),(synsem:CSynsem1)] ).
```

- Problem: Overgeneration
- Every kind of argument can occur in any position
- . . . e.g. Empty pronouns cannot occur everywhere

Another Implementation for Word Order

```
head_args_rule ##
(phrase, synsem:category:val:(subj:e_list, comps:e_list),
 daughters:(harg_struc, fdtr:Fdtr, hdtr:Hdtr, ndtrs:Ndtrs)) ===>

goal> (append(Fdtr,Ndtrs,(List,([_];[_,_];[_,_,_])))),
cats> Fdtr,
cat> (Hdtr,synsem:category:(head:verb,val:(subj:Subj, comps:Comps))),
cats> Ndtrs,
goal> (distribute(Fdtr,Ndtrs,Subj,Comps)).
```

- The first `goal` checks the length of the `Fdtr` and `Ndtrs` lists
- Use of `append` prevents infinite loops, deals with category `empty`
- Side effect: One word sentences like “kävellään” *one walks* analysed both as a word and a phrase

Predicate: Distribute

The predicate `distribute`

```
distribute(S,L,X1,Y1) if (extract(L,LE), extract(S,SE),
      append(LE,SE,LS), append(X1,Y1,XY), permuta(LS,XY)).
```

- checks that the `synsem` values of the `Fdtr` and `Ndtrs` lists are connected
- eventually permutates the same list as the two lists together
- does this with the help of `append`, `extract`, `permuta`
- `append` already in the signature

Predicate: Extract

The predicate `distribute` again:

```
distribute(S,L,X1,Y1) if (extract(L,LE), extract(S,SE),
      append(LE,SE,LS), append(X1,Y1,XY), permuta(LS,XY)).
```

The predicate `extract` sets a relation between the list of `signs` and the list of `synsem` values of that `sign`:

```
extract([],[]) if true.
extract(L1,L2) if (L1=(hd:(synsem:Synsem), tl:L1T),
       L2=(hd:Synsem, tl:L2T), extract(L1T,L2T)).
```

- applies when the head element of the second list matches with the `synsem` value of the head value of the second list
- also the rest of the list fits the predicate (second condition)
- always true for two empty lists (first condition)

Predicate: Permuta

The predicate `distribute` again:

```
distribute(S,L,X1,Y1) if (extract(L,LE), extract(S,SE),
      append(LE,SE,LS), append(X1,Y1,XY), permuta(LS,XY)).
```

The predicate `permuta` is true for two lists that have the same elements:

```
permuta([],[]) if true.
permuta([H|Tail],L) if (member(H,L), without(H,L,L2), permuta(Tail,L2)).
```

- Order of the elements on the lists is irrelevant
- Checks whether the head element of the first list is in the second list
- . . . and whether the rest of the list without the head element is a permutation of the second list

Predicate: Without

The predicate `permuta` again:

```
permuta([],[]) if true.
permuta([H|Tail],L) if (member(H,L), without(H,L,L2), permuta(Tail,L2)).
```

`without` is true when the deletion of the first occurrence of a given element from the first list results in the second list.

```
without(X,[X|Tail],Tail) if true.
without(X,[H|Tail1],[H|Tail2]) if without(X,Tail1,Tail2).
```

- If this element is the head element, the second list has to equal the rest of the first (first condition)
- Otherwise the head elements of both lists are ignored and the search continued in the rest of the list (second condition)

Conclusion

- Surveying this grammar made the process of writing grammars more concrete
- Finding the best approach for Finnish was surely time consuming
- The grammar is a good starting point for further implementation
- Well documented and compact, easily grasped
- Several versions on which to work
- B.A. Thesis idea: Implementing Discourse Functions

Thank You!