Appendix

A coordination algorithm

(1) INPUT

(a) An array of Types **S#n** = **[S#n$_1$ ... S#n$_n$]** such that $n \geq 3$
(b) An array of Types **S-co+fr** = **[S-co+fr$_1$ ... S-co+fr$_k$]** such that $k \leq n$
(c) A set of triples of Types **REDSET** such that $|\mathbf{REDSET}| = n - k$

{{ *The input is supposed to be the output of a Reduction Grammar as defined at the onset of chapter 3. Constituents are identified by their types and their positions in the input; the span is suppressed* }}

(2) OUTPUT

A set **COLEVELSET** of pairs of pairs of integers and arrays of **Types**

{{ *The set stores combinations of left and righthand coordination levels, each provided with an index marking the scope of coordination at that side. If this set ends up empty, no grammatical coordination has been established.* }}

(3) TYPE

A **Type** is a symbol in the set of linear types

{{ *A linear type is defined in chapter 3, section 2* }}

(4) FUNCTIONS

(a) *Decompose*

The function **Decompose** from **Types** to pairs of **Types** is the function h such that h(c) = <a, b> iff <a, b, c> is in **REDSET** and h(c) = <> (the empty pair) otherwise.

{{ *The function instantiates definition 207* }}

(b) *Insert_Replace*

The function **Insert_Replace** from pairs of **Types** <C_1, C_2> indices i and n-place arrays A to (n+1)-place arrays A′ is the function f such

that f(<C_1,C_2>,i, A) = A′ iff for all *j*, *j* < *i*, $A_j = A'_j$ and $A'_i = C_1$ and $A'_{i+1} = C_2$ and for all *j* > *i*+1, $A'_j = A_{j-1}$

{{ *The function is used for the construction of a string out of an other by replacing one particular constituent by its decomposition. The function embodies the Decomposition Protocol (257)* }}

(5) CONSTANTS

(a) *Axis*

AXIS is the lowest integer *i* such that **S-co+fr**$_i$ = **COORD**

{{ ***AXIS*** *identifies the relative position of the first coordinator in the input array* ***S-co+fr****. The algorithm, however, only provides a solution for strings with at one coordinator* }}

(b) *L-co+fr*

L-co+fr is the subarray **[S-co+fr**$_1$ **... S-co+fr**$_{AXIS-1}$**]** of **S-co+fr**

(c) *R-co+fr*

R-co+fr is the subarray **[S-co+fr**$_{AXIS+1}$ **... S-co+fr**$_n$**]** of **S-co+fr**

(6) VARIABLES: *S+co_l*, *S+co_r*

S+CO_L and **S+CO_R** are sets of pairs of integers and arrays of **Types**

{{ *The sets are stores for partial results in the search for coordination levels* }}

(7) PROCEDURE: *Find_Determinant*

output: **L_DET**, **R_DET**, **L_DET_POS**, **R_DET_POS**

{{ *This procedure fixes the Determinant, i.e., the central pair of constituents in the resolution of a coordination, and the positions of its members at* ***S-co+fr*** }}

(a) FUNCTIONS

(a.1) *Rleftc*

rleftc is the function that applied to a pair of **Types**, yields an integer such that
rleftc(y)(x/y) = **rleftc**(y)(x\z) = 0
rleftc(y)(x\y) = **rleftc**(y)(x) + 1

(a.2) *Rrightc*

rrightc is the function that applied to a pair of **Types**, yields an integer such that
rrightc(y)(x\y) = **rrightc**(y)(x\z) = 0
rrightc(y)(x/y) = **rrightc**(y)(x) + 1

{{ *These two functions instantiate the notion of Relative Complexity (245)* }}

(a.3) *Satisfy*

Satisfy is the function that applied to a pair of **Types** c_1 and c_2, yields the value **true** if c_1 satisfies c_2, and the value **false** otherwise

{{ *The function reflects the definition of potential satisfaction in clause (159)* }}

BEGIN

SET **Satpair** to the set of pairs of **Types** such that for each i and j <**L-co+fr**$_i$, **R-co+fr**$_j$> is in that set IFF **Satisfy(L-co+fr**$_i$**), R-co+fr**$_j$**)** = true

{{ ***Satpair*** *is the set of Satisfaction Pairs in* ***S-co+fr*** *as defined in chapter 3* }}

SET **Iospset** to that subset of **Satpair** such that <**L-co+fr**$_i$, **R-co+fr**$_j$> is in **Iospset** IFF for no $i' < i$ and no $j' > j$ OR for no $i' > i$ and $j' < j$ <**L-co+fr**$_{i'}$, **R-co+fr**$_{j'}$> is in **Satpair**

{{ *This clause determines the Innermost Outermost Satisfaction Pairs according to definition (228). Evidently,* ***Iospset*** *contains at least one and at most two members if* ***Satpair*** *is not empty* }}

SET **Func_Iospset** to the set of those **Types** that are the first or the second member of a pair in **Iospset** and that f-satisfy the other member of the pair.

{{ *Selects the f-satisfiers in* ***Iospset*** }}

SET **Inner_Iospset** to the set of those **Types L-co+fr**$_i$ and **R-co+fr**$_j$ such that for no $i' < i$ **L-co+fr**$_{i'}$ is the first member of a pair in **Iospset** and for no $j' > j$ **R-co+fr**$_{j'}$ is the second member of a pair in **Iospset**

{{ *Selects the innermost* ***Types*** *in* ***Iospset*** }}

SET **Comarker** to that **Type** that is in the intersection of **Func_Iospset** and **Inner_Iospset**

{{ *Determines the unique coordination marker, according to definition (239)* }}

SET **compl** to the integer such that
IF for some *i* **Comarker** = **L-co+fr**$_i$ and **Comarker** = x/y
THEN **compl** = **rrightc**(y)**(Comarker)**
ELSE if **Comarker** = x\y, then **compl** = **rleftc(y)(Comarker)**

{{ *Determines the relative complexity of* ***Comarker*** }}

SET **Comarker_sp** to that ordered subset of **Satpair** such that for all *i* and for all *j*
<**L-co+fr**$_i$, **R-co+fr**$_j$> is in **Comarker_sp** IFF either **Comarker** = **L-co+fr**$_i$ or **Comarker** = **R-co+fr**$_j$
AND
IF for some *i* **Comarker** = **L-co+fr**$_i$ THEN the set is ordered by decreasing *j* of **R-co+fr**$_j$ ELSE the set is ordered by increasing *i* of **L-co+fr**$_i$

{{ *Selects the relevant satisfaction pairs in an ordered way, from outermost to innermost* }}

SET **Det** to the **compl**-th pair of **Types** in **Comarker_sp**

{{ *Selects the determinant according to (247)* }}

SET **L_DET** and **R_DET** to the member of **L-co+fr** and the member of **R-co+fr** in **Det**, respectively

SET **L_DET_POS** to the integer *i* such that **L_DET** = **S-co+fr**$_i$

SET **R_DET_POS** to the integer *j* such that **R_DET** = **S-co+fr**$_j$

{{ *This fixation of the index of the members of the determinant with respect to the inputstring* ***S-co+fr*** *is motivated by the locality of coordination, theorem (256). The ordering will be* ***L_DET_POS*** < ***AXIS*** < ***R_DET_POS*** }}

END.

(8) PROCEDURE: *L_Compare*

input: a pair (two-place array) of **Types**: **L_Searcher**
an array of **Types**: **Images**
an integer **i**
an array of **Types**: **Co_String**
output: the set **S+CO_L** of pairs of integers and arrays of **Type**

{{ *The procedure is to compare decompositions at the left to potential coordination-images at the right. L_Searcher is a decomposition pair, occurring in a left-hand-side Co_String at a position marked by the integer. Images is the array of categories in R-co+fr, not yet imaged. The variable S-co_left is the storage for left coordinates and their peripheral strings. The integer marks the index of the left border. Storing the left coordinates separated from the right coordinates (see the procedure R_Compare below) is allowed by clause (259b). By the Locality of Coordination Lemma (256), different coordinations only differ "vertically", which means that further decomposition at one side does not affect established coordinates at the other* }}

BEGIN

IF **L_Searcher** is the empty pair OR **Images** is the empty array
THEN true

{{ *There is no further decomposition or there are no images left* }}

ELSE IF **L_Searcher**$_2$ = last**(Images)**

{{ *Following Stepwise Decomposition (261)* }}

THEN IF length(**Images**) = 1

{{ *cf. clause 259b* }}

THEN add <**i**, **Insert_Replace**(**L_Searcher**, **i**, **Co_String**)> to **S+CO_L**

{{ *Coordinate found, according to that clause* }}

AND
L_Compare(
Decompose(L_Searcher$_2$**)**,
Images,
i+1,
Insert_Replace(L_Searcher, i, Co_String))

{{ *The assignment of C-images is withdrawn, according to the Backtracking Statement (263), and the C-image itself is decomposed, in a search for new possible C-images properly contained in it. The procedure is re-entered with parameters L_Searcher =*

Decompose(L_Searcher$_2$), Images = Images, i = i+1, Co_String = Insert_Replace(L_Searcher, i, Co_String) }}

ELSE **L_Compare(**
Decompose(L_Searcher$_1$),
Images - last(**Images**),
i,
Insert_Replace(L_Searcher, i, Co_String))

{{ *If the right member of L_Searcher has a C-image there were it should be in Images but other images are not yet paired, the left member is called for decomposition, according to statement (259d)* }}

AND
L_Compare(
Decompose(L_Searcher$_2$),
Images,
i+1,
Insert_Replace(L_Searcher, i, Co_String))

{{ *Withdraws the assignment, backtracking as above* }}

ELSE **L_Compare(**
Decompose(L_Searcher$_2$),
Images
i+1,
Insert_Replace(L_Searcher, i, Co_String))

{{ *Backtracking is necessary if no proper image was found in Images* }}

END.

(9) PROCEDURE: *R_Compare*

output: a pair (two-place array) of **Types**: **R_Searcher**
an array of **Types**: **Images**
an integer **i**
an array of **Type** Co_String
output: the set **S+CO_R** of pairs of an integer and an array of **Types**

{{ *R_Compare operates as L_Compare, under symmetrical adaptions. The parameters of the two procedures are essentially disjoint in extension* }}

```
BEGIN

  IF R_Searcher or Images is the empty pair cq. the empty array

  THEN halt

  ELSE IF R_Searcher_1 = first(Images)

      THEN    IF length(Images) = 1

                  THEN    add <i, Insert_Replace(R_Searcher, i,
                          Co_String)> to the set S+CO_R
                          AND
                          R_Compare(
                                Decompose(R_Searcher_1),
                                Images,
                                i,
                                Insert_Replace(R_Searcher, i, Co_String) )

                  ELSE    R_Compare(
                                Decompose(R_Searcher_2),
                                Images - first(Images),
                                i+1,
                                Insert_Replace(R_Searcher, i, Co_String) )
                          AND
                          R_Compare(
                                Decompose(R_Searcher_1),
                                Images,
                                i,
                                Insert_Replace(R_Searcher, i, Co_String) )

      ELSE R_Compare(
                  Decompose(R_Searcher_1),
                  Images,
                  i,
                  Insert_Replace(R_Searcher, i, Co_String) )

  END.
```

(10) PROCEDURE: *Fix_Coord*
output: the sets of pairs of integers and arrays of **Types** **S+CO_L** and **S+CO_R**

{{ *The procedure takes care of Coordination Resolution (196) in case one or both members of the innermost satisfaction pair qualify for decomposition, as pointed at in the Coordination Lemma (250). It activates and steers the initial run of the procedures L_Compare and R_compare. The output stores the left and right coordinates found — see also L_Compare* }}

BEGIN

SET **R_Images** to the subarray [**R-co+fr**$_1$...**R-co+fr**$_{R\_DET\_POS - AXIS - 1}$] of **R-co+fr**
SET **L_Images** to the subarray [**L-co+fr**$_{AXIS - L\_DET\_POS - 1}$ ... **L-co+fr**$_{AXIS-1}$] of **L-co+fr**

{{ *The two variables are motivated by the Resolution Statement (259). They contain the categories in S-co+fr that are within the scope of coordination at S+co and have to be C-imaged by decomposition. One of them may be empty, in case a member of the determinant is adjacent to COORD* }}

IF **L_DET_POS** = **AXIS** - 1

{{ *Inspecting the position of the left-wing det member: is it adjacent to COORD?* }}

THEN **L_Compare(**
Decompose(L_DET),
R_Images,
L_DET_POS,
L-co+fr)
AND
add <**R_DET_POS**, **R-co+fr**> to **S+CO_R**

{{ *Following clause (250b): no decomposition at the right wing; the right coordinate is fixed* }}

ELSE IF **R_DET_POS** = **AXIS** + 1

{{ *Is the right member of det a neighbour to* ***COORD****?* }}

THEN **R_Compare(**
Decompose(R_DET),
L_Images,
R_DET_POS,
R-co+fr)

AND
add <**L_DET_POS**, **L-co+fr**> to **S+CO_L**

{{ *If so, the left wing is fixed for coordination and to the right decomposing and comparison takes place* }}

ELSE **L_Compare(**
Decompose(L_DET),
R_Images,
L_DET_POS,
L-co+fr)
AND
R_compare(
Decompose(R_DET)
L_Images,
R_DET_POS
R-co+fr)

{{ *Otherwise, there is simultaneous decomposition at the left and the right, according to (250c)* }}

END.

(11) PROGRAM

BEGIN

IF length(**L-co+fr**) = length(**R-co+fr**) = 1

{{ *A special case* }}

THEN IF **L-co+fr**$_1$ = s AND **R-co+fr**$_1$ = s

{{ *Most special; cf. statement (199)* }}

THEN add <1, **L-co+fr**> to **S+CO_L** AND add <1, **R-co+fr**> to **S+CO_R**

{{ *Coordination found, of course* }}

ELSE IF **L-co+fr**$_1$ =/= s AND **R-co+fr**$_1$ =/= s
THEN SET **COLEVELSET** to the empty set

{{ *If the two strings have each just one member and none is of type s, there cannot be any coordination according to statement (220)* }}

ELSE IF **L-co+fr**$_1$ = s
THEN **L_Compare(**
Decompose(L-co+fr$_1$**)**
R-co+fr,
1,
L-co+fr)
AND
add <1, **R-co+fr**> to **S+CO_R**

{{ *There cannot be a determinant. Yet, decomposition is called for; cf. statement 215. The right side is fixed* }}

ELSE **R_Compare(**
Decompose(R-co+fr$_1$**),**
L-co+fr,
1,
R-co+fr)
AND
add <1, **L-co+fr**> to **S+CO_L**

{{ *The former case reversed* }}

ELSE IF length(**L-co+fr**) = 1
THEN **L_Compare(**
Decompose(L-co+fr$_1$**),**
R-co+fr,
1,
L-co+fr)
AND
add <length(**R-co+fr**), **R-co+fr**> to **S+CO_R**

{{ *On behalf of statements (200), (215), (216), (219) and (220), theorem (250) forces us to apply decomposition to a single category and to fix the other string if it contains more than one member* }}

ELSE IF length(**R-co+fr**) = 1
THEN **R_Compare(**
Decompose(R-co+fr$_1$**),**
L_co+fr,
1,
R-co+fr)
AND
add <length(**L-co+fr**), **L-co+fr**> to **S+CO_L**

{{ *The same case, the other way around* }}

ELSE **Find_Determinant**
Fix_Coord

{{ *Normal business, finally* }}

SET **COLEVELSET** to the cartesian product of **S+CO_L** and **S+CO_R**

{{ *The mission is over* }}

END.