

# Lexicalizing Romanian Path in Nanosyntax

Carmen Florina Savu  
carmenflorinasavu@yahoo.com

June 2013

## Abstract

This paper analyzes directional expressions in Romanian in the framework of Nanosyntax (Starke 2009, 2011; Caha 2009; Pantcheva 2011), looking at specific details such as the shapes of the lexical items and the steps of the derivation. Following Pantcheva's (2011) analysis of Path expressions crosslinguistically, I assume the decomposition of the traditional Path head into several terminals, (Scale)>Route>(Scale)>Source>(Scale)>Goal, which different languages divide between morphemes in different ways. I give a detailed description of the nanosyntactic apparatus and the way it applies to Path specifically (as proposed in Pantcheva 2011). I then use the theory to analyze Path in Romanian, showing how the Path heads are distributed between morphemes in this language, and how the derivation proceeds.

## 1 Introduction

This paper looks at directional expressions in the Romanian language from a nanosyntactic perspective, building on Pantcheva's (2011) analysis of Path.

Nanosyntax (Starke 2009, 2011; Caha 2009; Pantcheva 2011) argues for fine-grained pictures of syntactic expressions, in which one morpheme may span over several of the very small terminals, which contain one feature each. Different languages have different ways of dividing the syntactic structure between morphemes, and in this paper I look at how Romanian divides its Path heads.

The outline of the paper is as follows. Section 2 contains a detailed presentation of Nanosyntax, illustrating its derivation mechanism and main principles. Section 3 presents the decomposition of Path expressions proposed in Pantcheva (2011). Sections 4 and 5 are devoted to Romanian Path constructions and their analysis. Section 6 concludes.

## 2 Nanosyntax

This section presents, in some detail, the core principles and workings of Nanosyntax (Starke 2009, 2011; Caha 2009; Pantcheva 2011), a syntactic framework currently in development chiefly at the University of Tromsø. In this paper I adopt and use the model presented in Pantcheva (2011), which will be the one described in what follows.

Nanosyntax falls within the cartographic approach, in the sense that it works with syntactic trees built from small, atomic features. Features do not 'bundle' into heads. As such, syntactic heads are very small, and there is a strict one-feature-per-head constraint. Since there are no heads with bundles of several features, the way to handle the frequent mismatch between the number of morphemes in a given expression and the number of terminals (features) it expresses is to consider that one morpheme may span over several terminals.

Features occur in a rigid hierarchy, which determines the way in which they are ordered. This is called the Functional Sequence, or *f-seq*. The only configurations in which features may be arranged are the ones that syntax can build, either by Merge, or, as we shall see, by Move. Thus, we get fine-grained pictures of syntactic expressions.

## 2.1 The Lexicon and Nanosyntax

In the nanosyntactic framework, Syntax builds lexical items. There is no Lexicon coming before syntax to feed it. Therefore, syntax is pre-lexical. Since syntax builds lexical items, morphemes come out as a simple reflection of how chunks of syntactic structure are stored in the Lexicon. Thus, it may be said that Nanosyntax puts syntax in the Lexicon. Rather than storing information about what features a certain lexical item contains by giving only an unordered set, the nanosyntactic Lexicon also gives information about the geometrical configuration of each item's features. That is, the Lexicon stores sub-trees. A stored morpheme (lexical item) consists of syntactic structure, which is obligatory, optionally paired with a phonological exponent, and conceptual (encyclopedic) content. Examples of toy lexical items are given below.

- (1) The structure of a lexical item:  
*morpheme*  $\Leftrightarrow$  <(phonological exponent), syntactic structure, (phonological content) >
- (2) a. *li*  $\Leftrightarrow$  </li/, A , ON > spanning over feature A  
 b. *la*  $\Leftrightarrow$  </la/, CP , IN > spanning over features A, B, and C
- $$\begin{array}{c}
 \diagup \quad \diagdown \\
 \text{C} \quad \text{BP} \\
 \diagup \quad \diagdown \\
 \text{B} \quad \text{A}
 \end{array}$$
- c. *lo*  $\Leftrightarrow$  </lo/, DP > spanning over feature D
- $$\begin{array}{c}
 \text{D}
 \end{array}$$

Notice that examples (2a-b) are listed with conceptual content, while (2c) is not. This means that (2a) and (2b) can only be used when building an expression belonging to the ON and IN encyclopedic series respectively. (2c), on the other hand, is not associated with a specific encyclopedic series, as it only contains a phonological exponent and formal features. Because of this, entry *lo* can be used in expressions of any encyclopedic series, while storing conceptual content in an entry will limit its range of use.

## 2.2 Spell-Out

In the stage before something is uttered, the speaker builds the syntactic structure of said utterance. Spell-Out is a matching operation that replaces a piece of the syntactic structure being built with a suitable lexical entry, meaning an entry whose lexical specification contains the node that the Spell-Out operation is inspecting for lexicalization. By Spell-Out, a node in the tree is supplied with the phonological content of the entry it is lexicalized with.

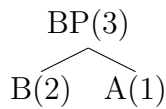
One of the crucial characteristics of Nanosyntax is that the Spell-Out operation targets terminal nodes, but also phrasal nodes. This happens in cases where one morpheme spells-out more than one terminal. Phrasal Spell-Out is a simple way of working with a Lexicon that stores mini-trees, given that the trees themselves will contain phrasal nodes.

In Nanosyntax, the syntactic structure (derivation) is built bottom-to-top by the operation External Merge, adding one feature at a time. The addition of a feature marks a new cycle. The Spell-Out operation is cyclical, meaning that after each External Merge (i.e. after the addition

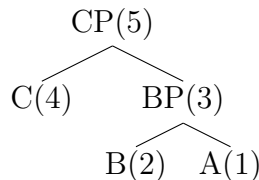
of each new feature), the Lexicon is consulted in search of a lexical item that can spell-out the new node(s) created since the last round of lexical access. The order in which nodes are inspected for lexicalization is bottom-to-top, right-to-left. This illustrated in (3) below.

(3) Order of lexicalization:

a. Cycle 1



b. Cycle 2



In (3), the first cycle contains the merger of B with A into BP. Going from right to left, the system inspects A first, then goes to the left to B, after which it targets BP. The system now has a memory of the outcome of lexical insertions in this lexicalization round. After this, the second cycle begins with the addition of C to the structure. Since the system knows the result of the Spell-Out operations that have taken place in the previous cycle, nodes A, B and BP do not have to be inspected again. Therefore, in this cycle the first targeted node is C, after which the system proceeds to the last node, namely CP.

The condition for a successful derivation is that all terminal nodes must be lexicalized at the end of every cycle. Having an unexpressed feature when a cycle ends makes the derivation crash.

(4) Cyclic Exhaustive Lexicalization (Pantcheva 2011: 115)

Every feature must be lexicalized at the end of every cycle.

In order to illustrate the condition, consider a derivation with a single cycle. A good example is a short derivation that ends at point (3a) above. It is obligatory to insert items at nodes A and B, but not at BP. This is because by the time the system arrives at node BP, all terminal nodes are lexicalized and the condition for a successful derivation is met at the end of the cycle.

A terminal node may be lexicalized in two ways. If an entry is inserted at said terminal node, the lexicalization is direct. Alternatively, a node is lexicalized if all its daughters are lexicalized. This is called lexicalization by inheritance. This is the situation of BP in (3a) before it is inspected. B and A, the daughters of BP, are both already lexicalized.

Therefore, according to the condition in (4), inserting lexical items at phrasal nodes directly is optional. They must, however, be lexicalized at least by inheritance. In this, these nodes contrast with terminals, which do need to be lexicalized directly.

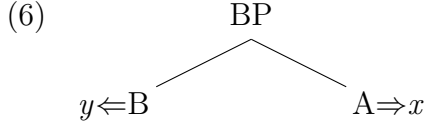
### 2.2.1 “Biggest Wins” (Starke 2009)

In order to illustrate this principle, consider the following toy lexical items.

- (5) a.  $x \Leftrightarrow < \quad A \quad >$   
 b.  $y \Leftrightarrow < \quad B \quad >$   
 c.  $z \Leftrightarrow < \quad BP \quad >$
- 
- ```

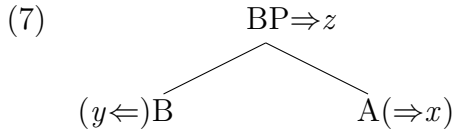
graph TD
    BP --- B
    BP --- A
  
```

The building of the syntactic structure starts with the merger of A and B into BP. After External Merge comes the first round of lexicalization. A is inspected first, and  $x$  is inserted. Then, B is lexicalized by  $y$ . The structure will look like in (6), and will be spelled-out as  $y + x$ <sup>1</sup>.



Notice that, at this point, both terminal nodes in the structure are expressed. The condition for a successful derivation is met, with BP lexicalized by inheritance. However, the Lexicon also contains item  $z$ , (spanning over features A and B), which contains the node BP. What “Biggest Wins” says is that direct lexicalization, if possible, is preferable to lexicalization by inheritance. This means that when BP is inspected and  $z$  is found to be a match, it is inserted, even though this lexical insertion is not necessary for the derivation to converge.

When  $z$  is inserted at BP, it overrides previous insertions at the daughter nodes, since  $z$  contains the entire tree, the outcome of A merged with B, and can therefore spell it out. The smaller lexical items  $x$  and  $y$  are no longer necessary. This is the essence of the “biggest wins” statement. The ‘bigger’ lexical entry  $z$  overrides the expression of the structure as  $y + x$ . The entire structure is spelled-out as  $z$  instead.



This principle leads to the general expectation that ‘bigger’, portmanteau items spanning over several features will override analytical expressions formed of ‘smaller’ pieces. A good example of this is the fact that the synthetic expression *went* overrides the analytical *\*go+ed*.

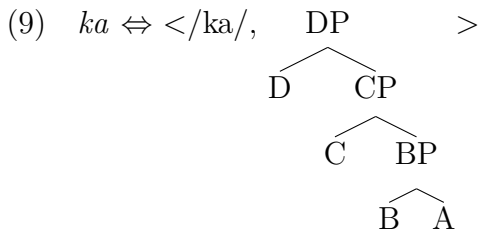
### 2.2.2 The Superset Principle

The Superset Principle states the criterion by which the system decides whether or not a lexical item is a match for insertion at a node.

- (8) The Superset Principle (Starke 2009; Caha 2009 in Pantcheva 2011) - to be revised  
A vocabulary item matches a node if its lexical entry is specified for a constituent containing that node.

In other words, the Superset Principle states that entries must be identical to, or contain the structures they lexicalize. An item that is specified for several features will contain a sub-tree. Such an item may be used to spell-out the entire sub-tree it is specified for, or any of the tree’s sub-constituents. This means that, in Nanosyntax, lexical items are over-specified, rather than under-specified.

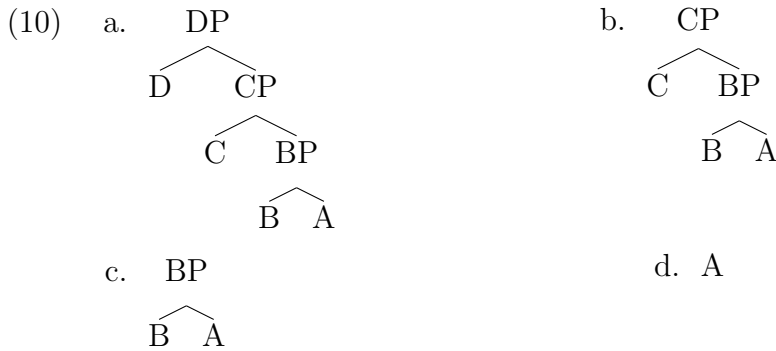
To illustrate the principle, consider the lexical item in (9).



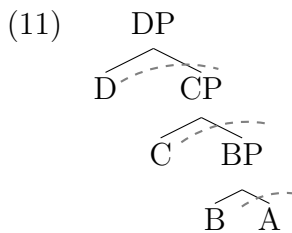

---

<sup>1</sup>Nanosyntax adopts Kayne’s Linear Correspondence Axiom, according to which asymmetric c-command maps into linear precedence.

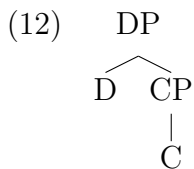
The entry *ka* can, by virtue of the Superset Principle, lexicalize any of the structures in (10):



(9) is a ‘big’ lexical item that spans over features A, B, C and D. By peeling off one layer at a time starting from the top, we get the structures that *ka* can spell-out.



It is important to start from the top when peeling the layers, as (11) shows, and to use the features in a morpheme starting from the bottom. A lexical item cannot be used to spell-out features from the top shells without using the lower shells. Therefore, if the structure in (12) needs to be lexicalized, *ka* cannot be used without A and B being in the structure, even though it contains the relevant features, because that would mean using the topmost layers of the entry.

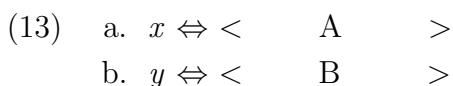


Given the Superset Principle, morphemes which span over two or more terminals may lexicalize more than one structure, as *ka* may lexicalize all the structures in (10). This is the source of syncretic expressions. Such expressions will be ambiguous between the different structures the item can be used to spell-out. For example, if *ka* is used, the hearer will not be sure if the syntactic structure contains up to feature D or only up to features C, B or A.

### 2.2.3 “Minimize Junk” - The Elsewhere Condition

Because of the Superset Principle, it is possible for a lexical item to spell-out structures smaller than said lexical item is specified for, as already explained at the end of the previous subsection. A consequence of this is that, from the point of view of the structure to be spelled-out, lexical entries will often contain superfluous features.

As already seen from the mini-Lexicon in (5) repeated below in (13), the same feature may be contained in several entries. For example, feature B is contained in both *y* and *z*.



$$\text{c. } z \Leftrightarrow < \begin{array}{c} \text{BP} \\ \swarrow \searrow \\ \text{B} \quad \text{A} \end{array} >$$

“Minimize Junk” is the criterion by which it is decided which lexical item to use in such a situation. The idea is to use the item that contains the smallest number of superfluous features.

- (14) When two lexical entries meet the conditions for insertion at a given node, the item with the fewest features not contained in the node gets inserted. (Pantcheva 2011: 127)

As an example, consider the lexical items in (13). The building of a syntactic structure starts by merging A and B into BP.

$$(15) \quad \begin{array}{c} \text{BP} \\ \swarrow \searrow \\ \text{B} \quad \text{A} \end{array}$$

When the lexicalization round begins, the first inspected node is A. Consulting the Lexicon brings up two matching lexical items.  $x$  and  $z$  both contain feature A. By the Superset Principle,  $z$  is eligible for insertion at node A as well. The system will decide between  $x$  and  $z$  by using “Minimize Junk”. Item  $z$  has one superfluous feature, B, while item  $x$  is identical to the node to spell-out, containing no superfluous features. Item  $x$  will win the competition and will be inserted.

There may not be an ‘ideal’ lexical item which will win the competition by being a perfect match. Suppose we have the two lexical items in (16) and the structure to be lexicalized in (17).

$$(16) \quad \begin{array}{ll} \text{a. } ki \Leftrightarrow < \begin{array}{c} \text{EP} \\ \swarrow \searrow \\ \text{E} \quad \text{DP} \\ \quad \swarrow \searrow \\ \quad \text{D} \quad \text{CP} \\ \quad \quad \swarrow \searrow \\ \quad \quad \text{C} \quad \text{BP} \\ \quad \quad \quad \swarrow \searrow \\ \quad \quad \quad \text{B} \quad \text{A} \end{array} > & \text{b. } ka \Leftrightarrow < \begin{array}{c} \text{DP} \\ \swarrow \searrow \\ \text{D} \quad \text{CP} \\ \quad \swarrow \searrow \\ \quad \text{C} \quad \text{BP} \\ \quad \quad \swarrow \searrow \\ \quad \quad \text{B} \quad \text{A} \end{array} > \end{array}$$

$$(17) \quad \begin{array}{c} \text{CP} \\ \swarrow \searrow \\ \text{C} \quad \text{BP} \\ \quad \swarrow \searrow \\ \quad \text{B} \quad \text{A} \end{array}$$

Both (16a) and (16b) can spell-out the structure in (17). Neither of them is a perfect match. By “Minimize Junk”, (16b) wins the competition because it has just one extra feature, whereas (16a) has two.

To sum up, there likely will be junk. The idea is to minimize it.

## 2.3 Movement

In Nanosyntax, movement is driven by the need to create a configuration suitable for an entry to spell-out a certain node. If a node needs to be spelled-out by an entry that does not contain the lower layers of the structure it is inserted in, those lower layers will have to evacuate, otherwise the item cannot be used.

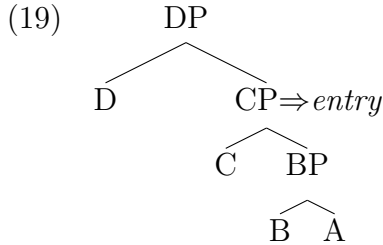
To illustrate, consider the lexical entry in (2c), repeated below:

$$(18) \quad lo \Leftrightarrow \langle /lo/, DP \rangle$$

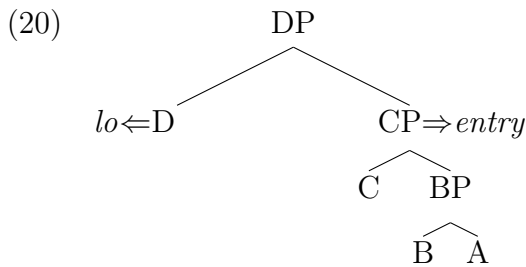
$$\quad \quad \quad |$$

$$\quad \quad \quad D$$

Suppose now that the derivation has reached the stage where CP has been spelled-out and D is merged:



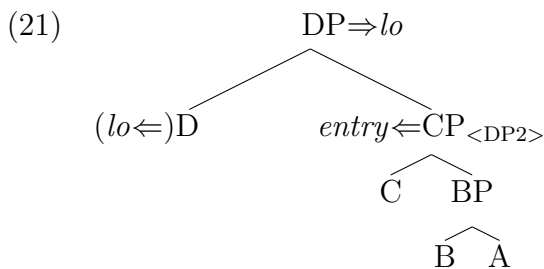
The system now inspects node D for lexicalization and entry *lo* is found to be a match:



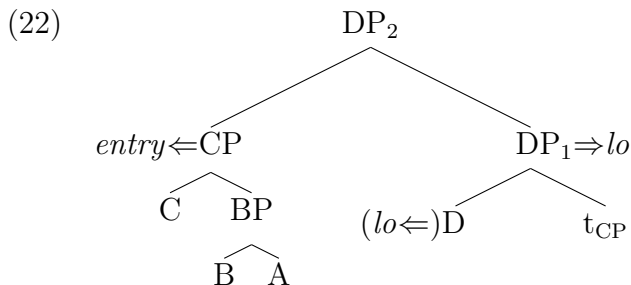
The next inspected node is DP, and *lo* is found to be a match again. However, *lo* does not contain CP and everything CP itself contains. *lo* cannot spell out a structure it does not contain in its lexical specification, so CP stands in the way of *lo* being inserted at node DP.

In order for *lo* to be able to spell-out DP, CP needs to move out of the way. It is, therefore, evacuated to a position just above the node that triggered the evacuation. In this case, CP needs to move right above DP, creating a new DP node, a new segment in the D category.

Entry *lo* will agree to spell-out DP, provided that the ‘promise’ is made that CP will move out as soon as possible. This means that CP is marked for extraction to DP<sub>2</sub> while *lo* gets inserted at the original DP node.



After the evacuation of CP, the structure will be spelled-out as *entry* + *lo* and will look as follows:



Notice that, when CP moves, it leaves a trace behind, yet *lo* still gets inserted at DP<sub>1</sub>. Traces, therefore, do not affect the Spell-Out system and are ignored when checking if an item is a match for insertion at a node. In light of this, the matching condition of the Superset Principle in (8) is reformulated as in (23).

- (23) A vocabulary item matches a node if its lexical entry is specified for a constituent containing that node, ignoring traces. (Pantcheva 2011: 139)

It is worth noticing, at this point, that *lo* is inserted at DP even though the node was lexicalized by inheritance before the system inspected it (see (20) above). *lo* still gets inserted, despite the movement required to reach the appropriate configuration. From this, we can conclude that direct lexicalization with movement is preferable to lexicalization by inheritance without movement, if a lexical item that can be inserted exists. We may, thus, expect that the lexicalization process analyses every node constructed in the derivation separately, attempting direct lexicalization. It is not checked whether or not this is strictly necessary to make the derivation converge. If direct lexicalization is possible, the system performs it. If direct lexicalization is possible, but requires movement, the system agrees to move.

As the steps are sketched above, the system marks CP for extraction and inserts *lo* at DP before actually performing the movement. Metaphorically, we might say that *lo* receives a promise that CP will evacuate and, based on that promise, the lexical item agrees to spell-out DP. The system will keep its promise a little later, at the beginning of the next cycle, right before the merger of the next feature.

The story gives us the following structure of a cycle:

- Potentially vacuous movement operation (Internal merge) of material marked for extraction in the previous cycle
- Addition of a new feature to the derivation (External Merge)
- Lexicalization round of all nodes not previously targeted for lexicalization, including the ones created by the movement operation at the beginning of the cycle
- Potentially vacuous marking for extraction of material to be evacuated in the next cycle

## 2.4 An example of an (abstract) derivation

For the purpose of illustration, this subsection goes through the details of an abstract derivation, using toy lexical items and making use of all the principles and mechanisms outlined in the previous subsections.

Consider a Lexicon containing the following lexical items.

- (24) a.  $x \Leftrightarrow \langle /za/, \quad A \quad \rangle$   
       b.  $y \Leftrightarrow \langle /ze/, \quad B \quad \rangle$   
       c.  $z \Leftrightarrow \langle /zi/, \quad BP \quad \rangle$
- $$\begin{array}{c} \diagup \quad \diagdown \\ B \quad A \end{array}$$
- d.  $v \Leftrightarrow \langle /zo/, \quad C \quad \rangle$   
       e.  $w \Leftrightarrow \langle /zu/, \quad EP \quad \rangle$
- $$\begin{array}{c} \diagup \quad \diagdown \\ E \quad DP \\ | \\ D \end{array}$$



The system starts the first cycle by merging A and B. The first step in a cycle is not performed, as the structure is just beginning to be created and there is nothing to move.

After A and B are merged into BP, the structure is a target for lexicalization. Node A is inspected. Entries  $x$  and  $z$  match.  $x$  is a perfect match and is chosen because of “Minimize Junk”, since it contains no superfluous features.

Next, the system inspects node B.  $y$  and  $z$  match, as they contain node B in their lexical specification. By the same “Minimize Junk” principle,  $y$  is chosen as the winner.

The following step is to lexicalize BP, even though this node is already lexicalized by inheritance and at this point we have a convergent derivation. Item  $z$  is inserted this time, overriding the items inserted at the daughter nodes. The structure is spelled-out as  $z$ , meaning  $/zi/$  at this point.

The next step would be to mark for extraction material that needs to move in the next cycle. Since no movement was necessary to insert  $z$  at BP, this step is skipped. This first cycle is sketched in (25) below.

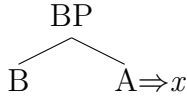
(25) Cycle 1:

a. merge A and B

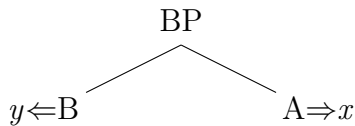


b. lexicalization round

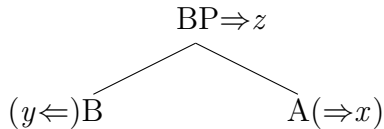
i. inspect node A, insert  $x$



ii. inspect node B, insert  $y$ ; the structure is spelled-out as  $y + x$



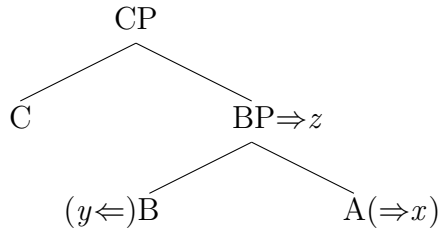
iii. inspect node BP, insert  $z$ ; the structure is spelled-out as  $z$



The second cycle of our derivation begins with the merger of C, as there is no movement from the previous cycle scheduled to be performed. The next step is the lexicalization round, in which C is the first node targeted. Entry  $v$  gets inserted, as it is the only entry containing feature C. The system proceeds to node CP and nothing is inserted, as the Lexicon contains no entry which has the CP node. The derivation still converges, as CP is lexicalized by inheritance after the spell-out of its daughter nodes, C and BP. At this stage, the structure is spelled-out as  $v + z$  ( $/zozi/$ ). Again, nothing gets marked for extraction, so the final general step of a cycle is not performed in this case.

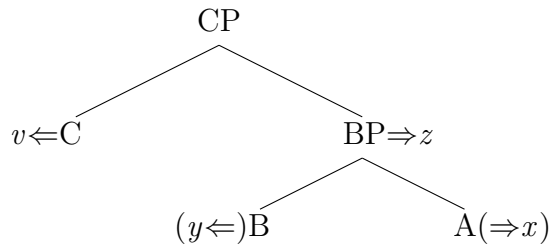
(26) Cycle 2:

a. merge C



b. lexicalization round

i. inspect C, insert  $v$ ; the structure is spelled out as  $v + z$



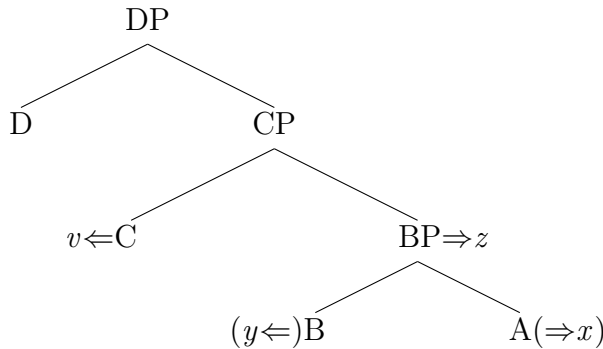
ii. inspect CP, nothing is inserted; the derivation is convergent

The third cycle begins with the addition of the feature D to the derivation, after which the structure is shipped to the Lexicon for Spell-Out, during the lexicalization round of this cycle. Node D is inspected and  $w$  is chosen despite having an extra feature, since it is the only one that has the node D. The Superset Principle makes it possible for  $w$  to be inserted, making use of its lowest layer. The structure is spelled-out as  $w + v + z$ .

The next inspected node is DP.  $w$  is again a match, a possibility granted by the Superset Principle. It is inserted, but under the condition that CP moves out, since  $w$  is not specified for CP. CP is, thus, marked for extraction.

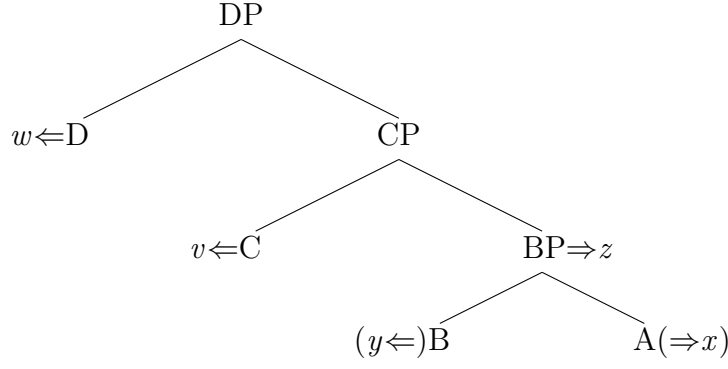
(27) Cycle 3:

a. merge D

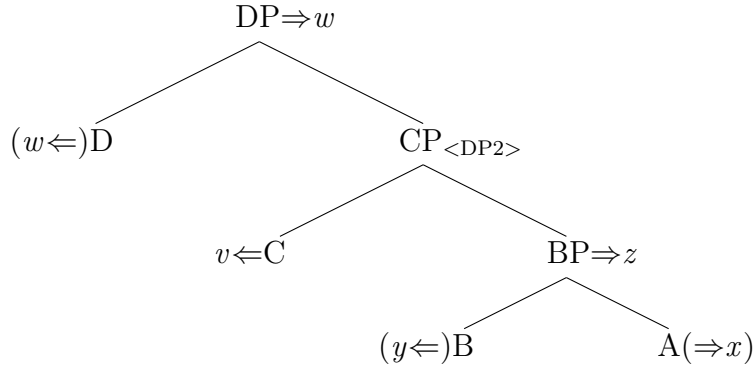


b. lexicalization round

- i. inspect D, insert  $w$ ; the structure is spelled-out as  $w + v + z$



- ii. inspect DP, insert  $w$  and mark CP for extraction

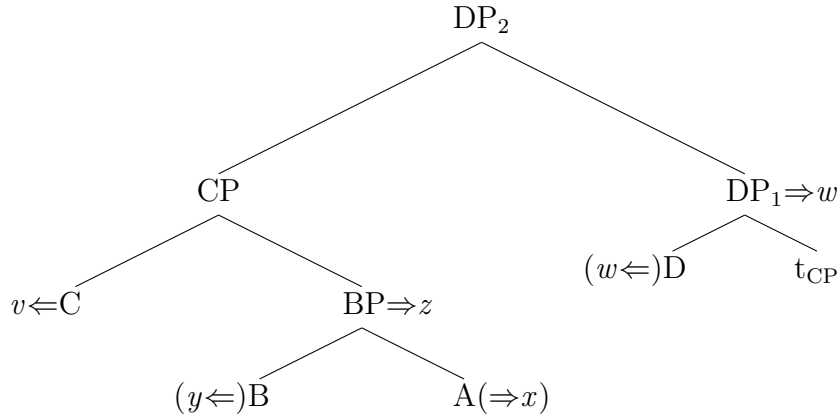


With the beginning of the fourth cycle, we have the opportunity to see the first step of a cycle, Internal Merge, taking place. The cycle begins with the performance of the evacuation ‘promised’ in the third cycle, CP to  $DP_2$ , thus creating a new node directly above the node that triggered the extraction,  $DP_{(1)}$ .

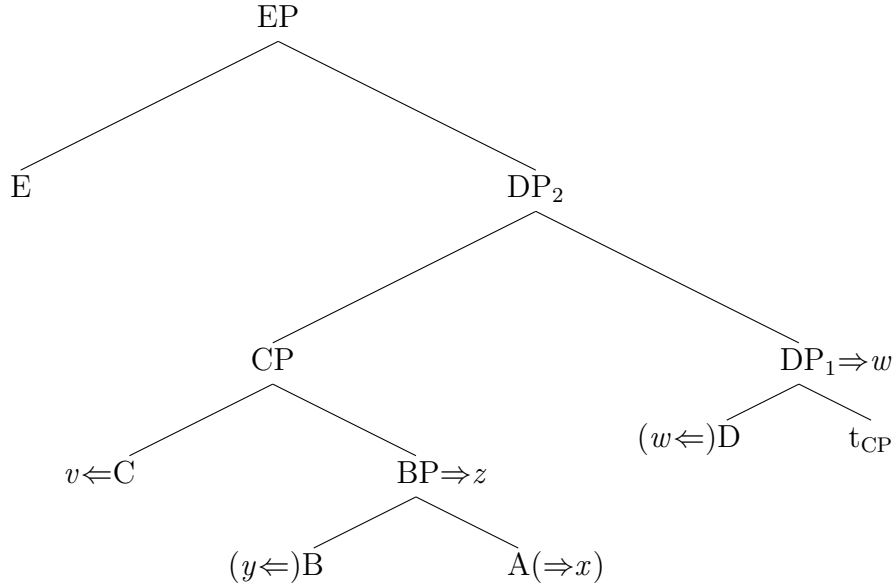
The following step is to merge the next feature, E.

(28) Cycle 4 (first steps):

- a. move CP to a position right above DP, creating node  $DP_2$



- b. merge E

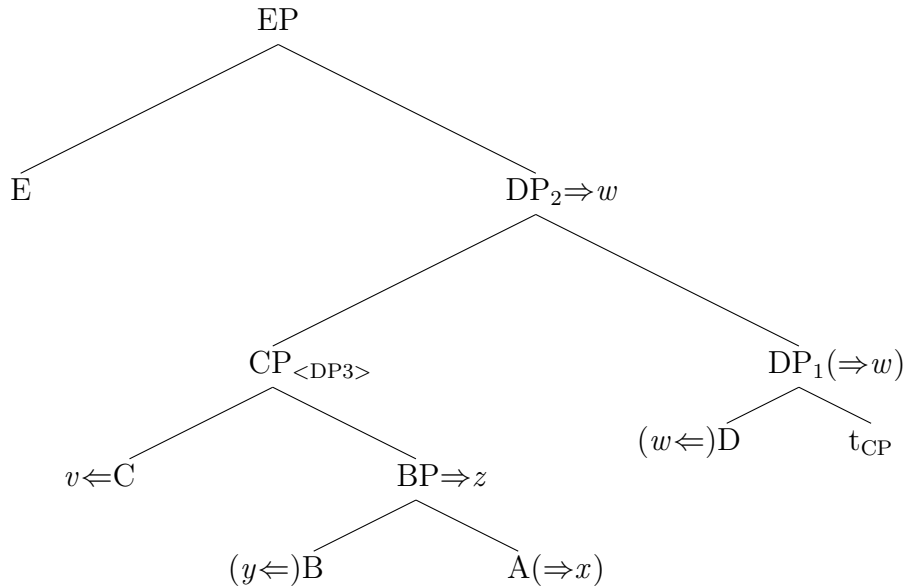


After this External Merge operation, the system proceeds to the lexicalization round. Remember that, in every cycle, the nodes targeted are the ones that have not previously been inspected, in a bottom-to-top, right-to-left order. Therefore, the first targeted node now is  $DP_2$ , the rightmost, bottommost node created since the last lexical access. Node  $DP_2$  belongs to the D category, since it contains a D feature, and the only entry suitable to spell-out this node is  $w$ . However,  $DP_2$  has CP within it, so CP needs to evacuate and it is marked for extraction at  $DP_3$ .

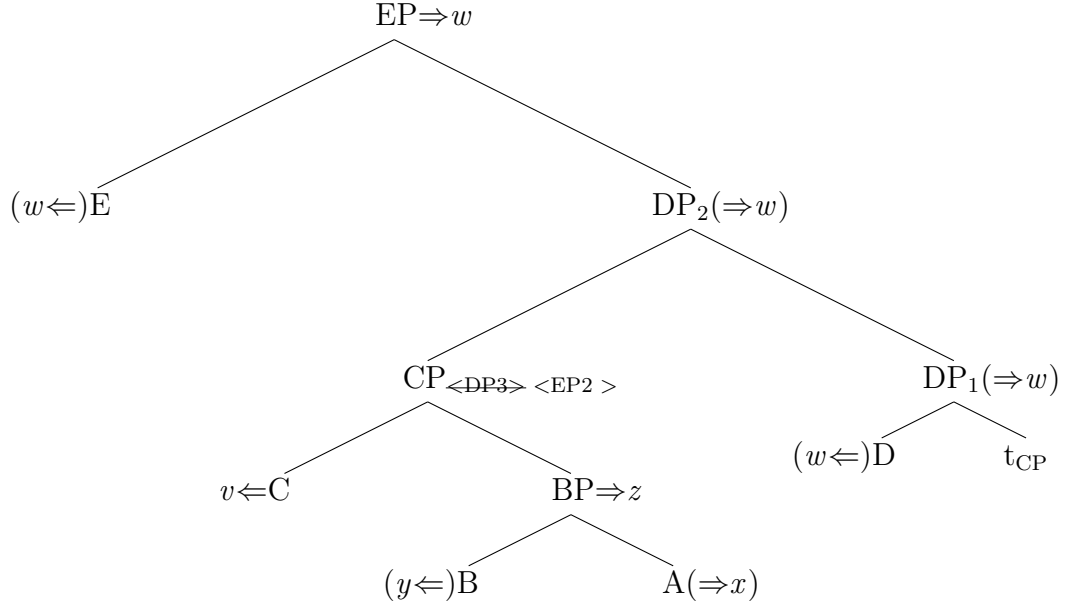
Node E is inspected next.  $w$  is a match and is inserted. The system proceeds to node EP and  $w$  is again found suitable. CP is, though, still in the way. If it were to be moved to  $DP_3$ , as previously marked, it would still be contained in EP phrase, keeping  $w$  from spelling-out the EP node. We thus have a situation in which CP must move to a position immediately above EP. The  $DP_3$  index is deleted and is replaced with one appropriate for throwing CP out from under the umbrella of EP. This movement is the one that will eventually take place in the next cycle.

(29) Cycle 4 (continued): lexicalization round

a. inspect  $DP_2$ , insert  $w$  and mark CP for extraction to  $DP_3$



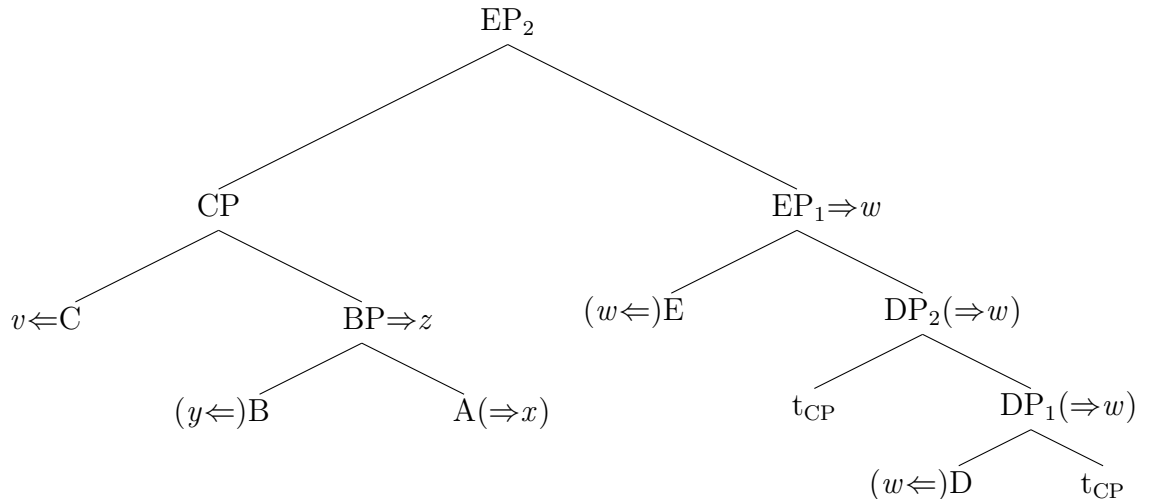
- b. inspect E, insert  $w$ ; inspect EP and mark CP for extraction to  $EP_2$



This is the end of Cycle 4 and the end of the addition of new features to the structure. Notice that the system still owes a movement operation to entry  $w$ . CP has to be evacuated because  $w$  cannot deal with it. However, as previously mentioned, a new cycle is marked by the addition of a new feature. Since the system has to ensure CP moves out from the E category, a ‘defective’ cycle will still have to occur, in which the movement of CP is performed.

(30) Cycle 5:

- a. move CP to  $EP_2$

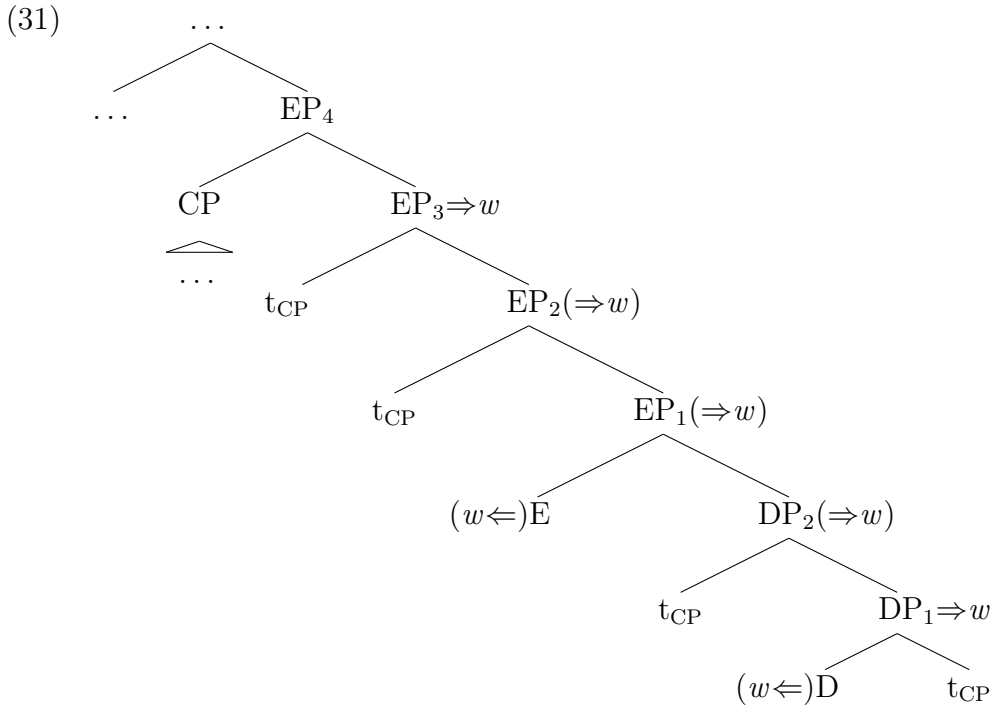


- b. no new feature is merged; no new lexicalization round; the derivation converges and the structure is spelled-out as  $v + z + w$ , meaning /zo zi zu/.

The defective Cycle 5 is triggered by the need to get CP out of EP. This cycle contains only the first of the general steps of a cycle listed in the previous subsection. It is the adding of a new feature that triggers lexical access. Since no new feature is added, the Lexicon is not

accessed in Cycle 5 and  $EP_2$  will not be inspected for lexicalization, even though it is a newly created node. Would CP in the E category not be a problem for the derivation, since entry  $w$  does not contain it? As long as no attempt is made for lexical insertion at the node, movement is not induced. The derivation is still successful, since  $EP_2$  is lexicalized by inheritance.

Besides the lack of a new feature, there is a practical reason for which it would not be a good idea to target  $EP_2$  for lexical insertion. Consider what would happen if there were a new round of lexical access after the creation of  $EP_2$ . The matching entry would be  $w$ , the only item with the E feature.  $w$  would get inserted, but CP would have to move and would be marked for extraction to  $EP_3$ . This need for extraction would trigger the occurrence of a new defective cycle like Cycle 5. The defective Cycle 6 would begin with the movement of CP to  $EP_3$ , after which  $EP_3$  would be sent to the Lexicon for Spell-Out. Entry  $w$  would be chosen yet again, with CP standing in the way and being marked for evacuation to  $EP_4$ , a marking which would create the need for defective Cycle 7. Cycle 7 would, in turn, be identical to Cycle 6 and would trigger Cycle 8, and so on. We would have a never-ending derivation.



To conclude, lexical access is triggered by merging a new feature to the structure, while a new cycle can also be triggered by the need to move material previously marked for extraction.

### 3 The Nanosyntax of Path (Pantcheva 2011)

With this section, we start moving away from theoretic principles and mechanisms and going towards applying the theory sketched in Section 2 to directional expressions. Specifically, this third section presents the decomposition of the Path head proposed by Pantcheva (2011) and how different types of Path are built.

#### 3.1 The classification of Paths

Before going into the decomposition of Path syntax, let us first see what kinds of Paths there are. According to Pantcheva (2011), Paths can be with/without:

- transition
- orientation
- delimitation

Paths with transition can be:

- |                                                        |                           |                 |
|--------------------------------------------------------|---------------------------|-----------------|
| • Goal-oriented: e.g. <i>to</i> the house              | $[ 0 - - - - + + + + 1 ]$ | } non-delimited |
| • Source-oriented: e.g. <i>from</i> the house          | $[ 0 + + + + - - - - 1 ]$ |                 |
| • Route (non-oriented): e.g. <i>past</i> the house     | $[ 0 - - + + + + - - 1 ]$ |                 |
| • Goal-oriented: e.g. <i>up to</i> the house           | $[ 0 - - - - - - - + 1 ]$ | } delimited     |
| • Source-oriented: e.g. <i>starting from</i> the house | $[ 0 + - - - - - - - 1 ]$ |                 |

Paths without transition can be:

- |                                                      |                           |                 |
|------------------------------------------------------|---------------------------|-----------------|
| • Goal-oriented: e.g. <i>towards</i> the house       | $[ 0 - - - - - - - - 1 ]$ | } non-delimited |
| • Source-oriented: e.g. <i>(away) from</i> the house | $[ 0 - - - - - - - - 1 ]$ |                 |
| • Route (non-oriented): e.g. <i>along</i> the river  | $[ 0 + + + + + + + + 1 ]$ |                 |

This classification makes the distinction between transitional and non-transitional Paths. Paths in which, at some points, the locations of the Figure and the Ground coincide, while at other points they do not, are transitional Paths. In non-transitional Paths, the locations either do not coincide at any point in the Path, or they coincide at all points, hence the lack of transition. Thus, we have two kinds of Route Paths, one in which the Figure is first not at the Ground, then at the Ground, then again not at the Ground (*past* the tree), and another one in which the Figure is at the Ground at all points in the Path (*along* the river).

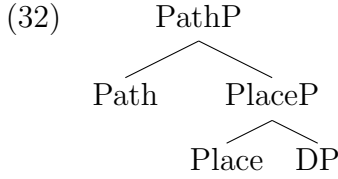
In this paper, I leave out delimited Paths and restrict the discussion to non-delimited ones only, both transitional and non-transitional<sup>2</sup>.

---

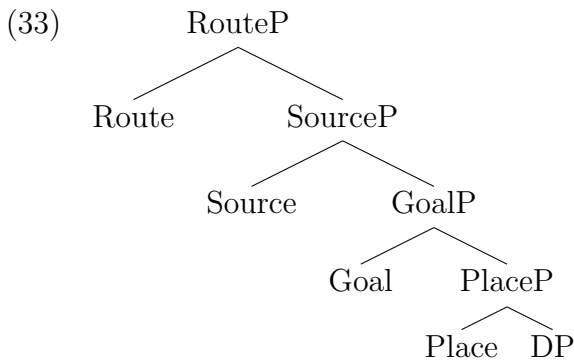
<sup>2</sup>Delimited Paths differ from their non-delimited counterparts in that a delimited Path restricts the number of points in which the locations of the Figure and the Ground coincide. For example, a non-delimited Goal Path may continue once the Ground has been reached, while its delimited counterpart is ended at this point. Syntactically, Pantcheva (2011) takes a delimited Path to be build by adding the Bound feature to a non-delimited Path. For reasons of simplicity, this paper does not discuss this head and the Paths it derives.

### 3.2 The decomposition of the Path head - building different types of Path

The traditional Path head builds a directional expression by taking a locative construction as its complement ((32)).



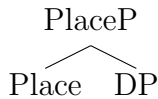
In the analysis in Pantcheva (2011) and adopted here, this traditional Path head is decomposed into three heads which are hierarchically ordered and in a containment relationship. These three heads are Route>Source>Goal.



Different types of Path are built incrementally, one on top of the other, and correspond to different sizes of the syntactic structure in (33). This is illustrated in what follows<sup>3</sup>.

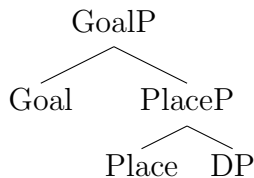
A locative construction is built by adding the Place head above the DP.

(34) Locative construction:



A Goal Path is built by adding the Goal head to a locative construction.

(35) Goal Path:



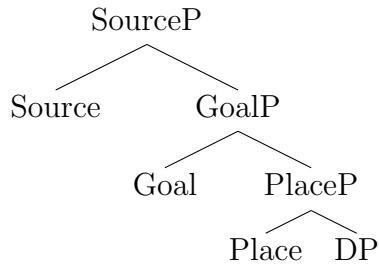
A Source Path is constructed by adding the Source head to a Goal structure.

---

<sup>3</sup>The trees in (35-37) give the structures for transitional Paths.

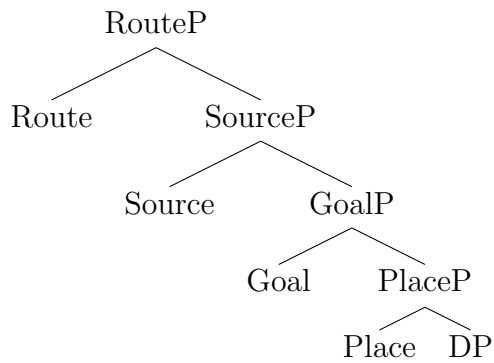


(36) Source Path:



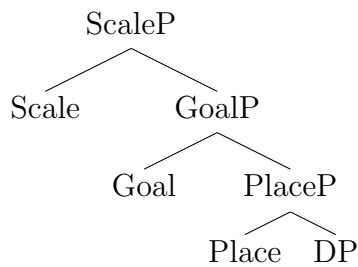
Finally, a Route head takes the Source Path as its complement and delivers a Route Path:

(37) Route Path:

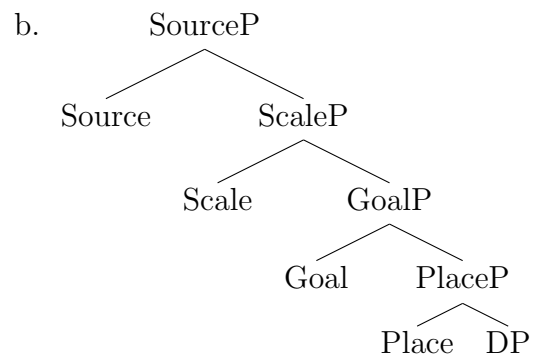
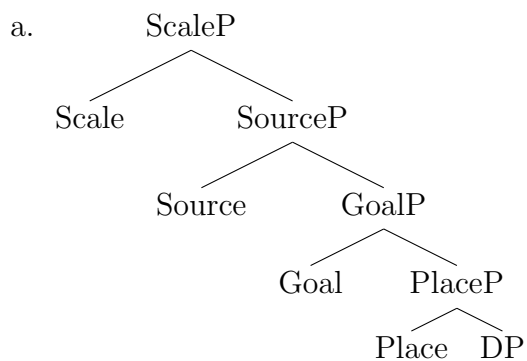


Non-transitional Paths are built by adding the Scale head to the corresponding transitional Path. This Scale head does not have a fixed position in the structure and may appear either on top of Goal, Source or Route.

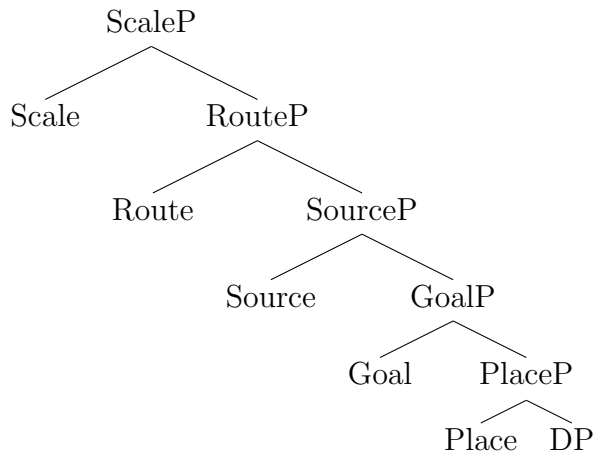
(38) Non-transitional Goal Path:



(39) Non-transitional Source Path:



(40) Non-transitional Route Path:



The function of each of these heads is as follows:

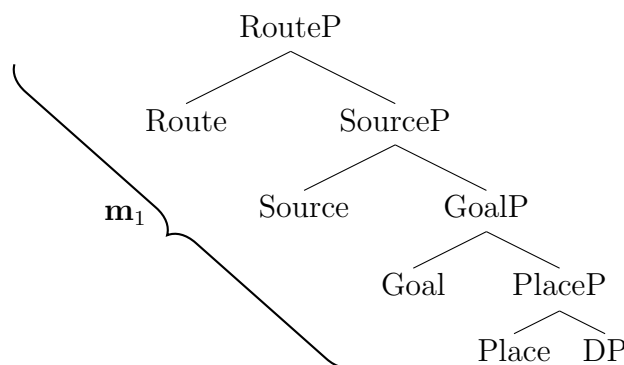
- **Place** – introduces a spatial region
- **Goal** – introduces a transition to the spatial region
- **Source** – reverses the orientation of the GoalP in its complement position
- **Route** – introduces a second transition (– to +) in the SourceP
- **Scale** – cuts out the transition, selecting either the negative interval (in Goal and Source Paths) or the positive one (in Route Paths) and making the newly selected interval the new Path; it also imposes an ordering on the points of the new Path such that the closer a point was to the positive points on the complement Path, the shorter the distance between Figure and Ground at that point

Given the nanosyntactic framework, where morphemes may span over one or several terminals, there will be various ways to divide the syntactic structure among morphemes. Thus, some languages may use morphemes that syncretize two or more features, while other languages may use more analytical expressions, in which each feature has its own terminal in Syntax. It is in these analytical-type languages that the incremental building of Paths may be observed, as morphemes will stack one upon the other when constructing the ‘bigger’ Paths.

(41) shows possible ways of partitioning a transitional Route Path. There are four heads in the internal structure of such a Path, which can be grouped in various ways.

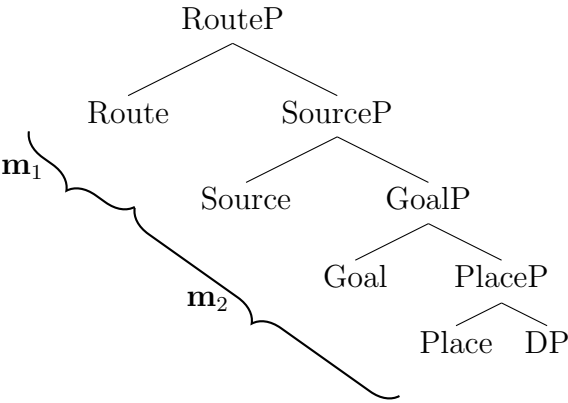
(41) Logically possible ways of expressing a transitional Route Path:<sup>4</sup>

a.

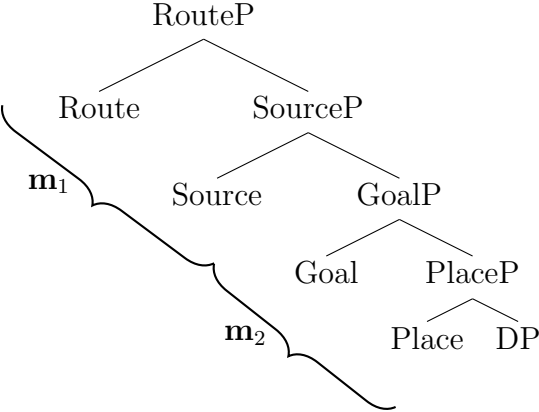


<sup>4</sup>Lexical items may overlap, in that the same feature may be found in several entries in the same language.

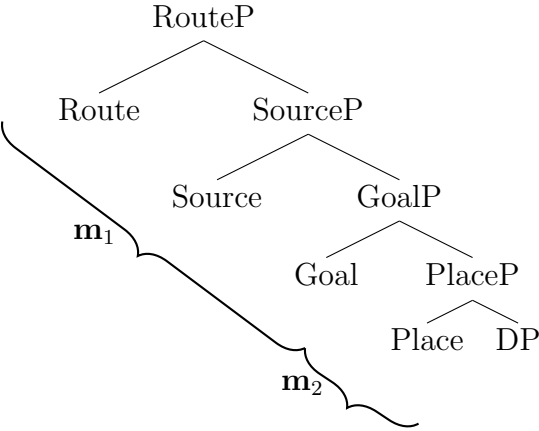
b.



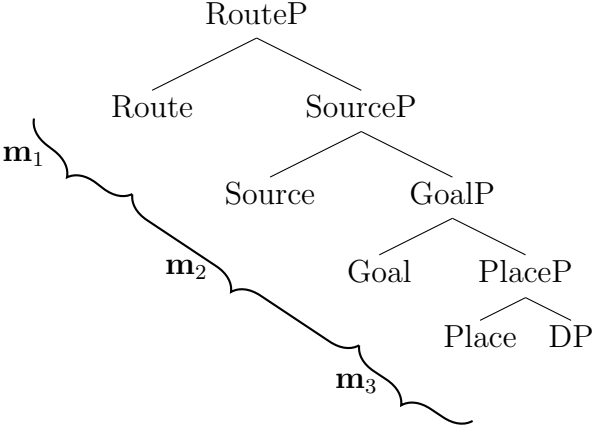
c.



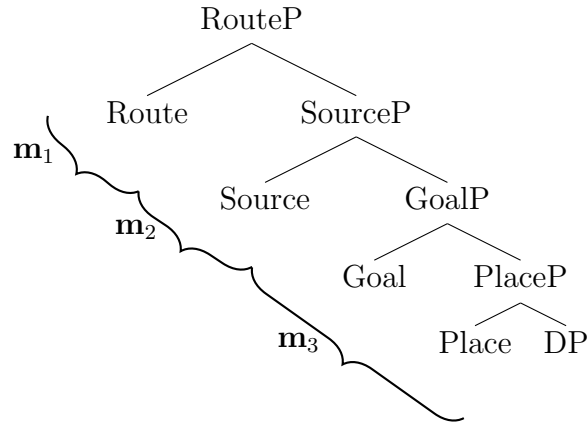
d.



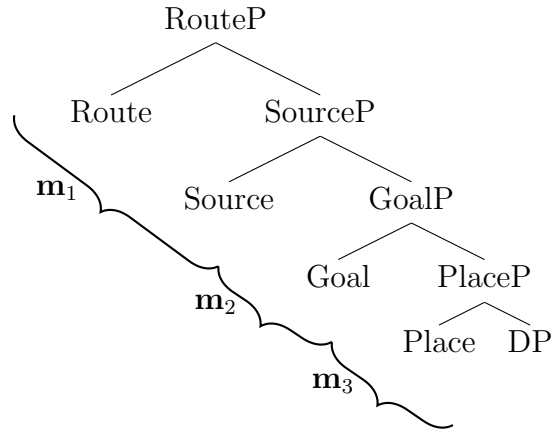
e.



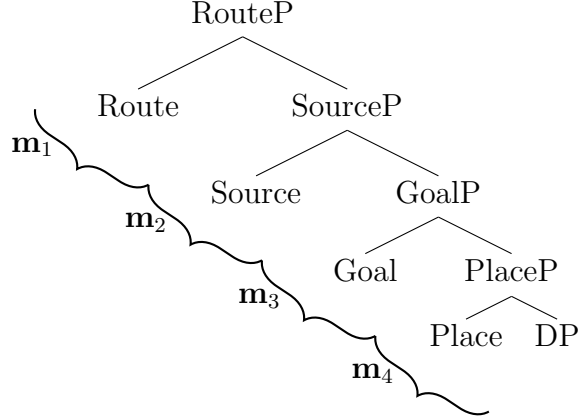
f.



g.



h.

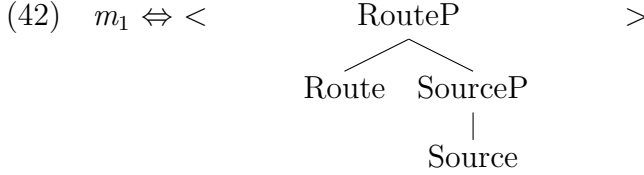


Nanosyntax contends that the *f-seq* is universal. This means that a certain Path structure will contain the same heads in any language. What differs is how each language chooses to partition and express said structure. Since syntactic expressions are expressed using lexical items, how each language spells-out a structure depends on what lexical items the language has at its disposal. As Starke (2011) argues, language variation will be reduced to the sizes of lexically stored trees.

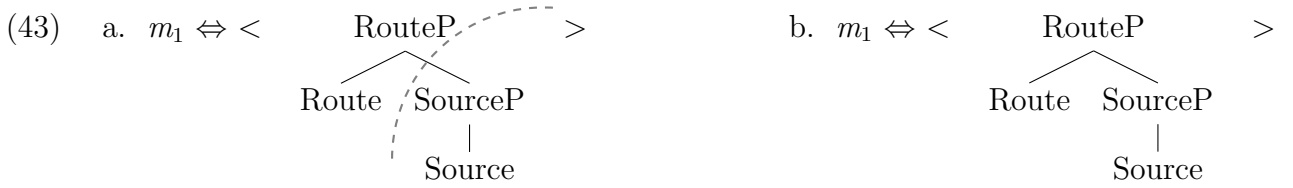
Looking at the trees in (41) above, we see logical possibilities for languages to express the same syntactic structure, that of a transitional Route Path. One language may choose to split the structure into two (41c), with one lexical item containing Route and Source and the other containing Goal and Place. Another language may choose (41h), in which each feature has its own separate morpheme. Path expressions in a language going the (41h) way will turn up as analytical. It is in such a language that it is easy to discern the heads in a Path structure, as morphemes will be stacked and clearly discernible.

Languages that take the (41c) way will be more opaque. In (41c)  $m_1$  is syncretic between Route and Source, while  $m_2$  is syncretic between Goal and Place. This means that when  $m_1$  is uttered, the expression may be either a locative one or a goal-of-motion one. Hence, ambiguity. Syncretism means ambiguity because of the Superset Principle and its emergence is explained in what follows.

Consider  $m_1$  in (41c). The morpheme spans over Route and Source. Therefore, these are the features it will contain.

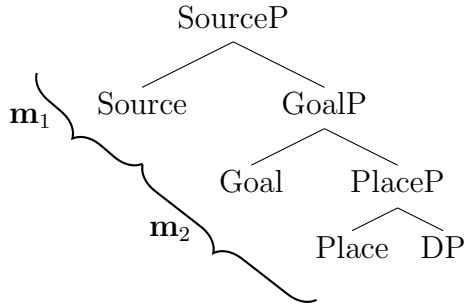


By virtue of the Superset Principle,  $m_1$  can use either only its bottommost layer, or both layers, meaning its entire lexical specification.

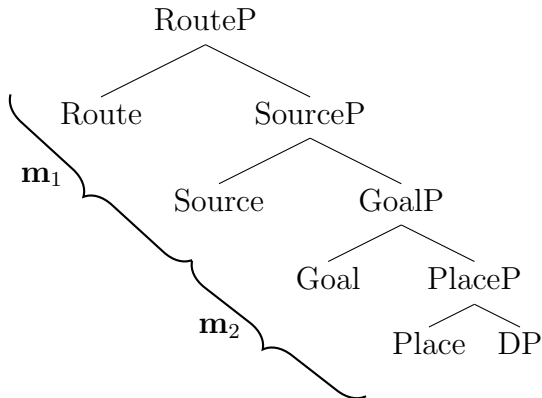


In both the following structures,  $m_1$  is uttered. Using  $m_1$  signals that at least Source is contained in the expression. But Route may be as well. Thus, we have a case of structural ambiguity and context will need to force one interpretation or the other.

(44) a. Source Path



b. Route Path



## 4 Expressing Path in Romanian

In this section we go to the specifics of Romanian directional expressions and towards a nanosyntactic analysis thereof.

Romanian uses prepositions to lexicalize Path expressions. In this paper we look at four series of conceptual content: IN, ON, AT, and UNDER<sup>5</sup>.

(45) The IN series

a. **(Place)**

Citesc în cameră.  
read.1SG ÎN room

‘I am reading in the room’

b. **(Goal – transitional)**

Elevii vin în clasă.  
students.DEF come.3PL ÎN classroom

‘The students are coming into the classroom’

c. **(Source – transitional)**

Leul a venit din pădure.  
lion.DEF have.3SG come DIN forest

‘The lion came out of the forest’

d. **(Route – transitional)**

Ana tocmai a trecut prin hol.  
Anna just have.3SG passed PRIN hallway

‘Anna just passed through the hallway’

e. **(Goal – non-transitional)**

Leul pășeste (în)spre/către pădure.  
lion.DEF step.3SG (ÎN)SPRE/CĂTRE forest

‘The lion is stepping towards the forest.’

f. **(Source – non-transitional)**

Leul a venit dinspre pădure.  
lion.DEF has.3SG come DINSPRE forest

‘The lion came from the direction of the forest’

g. **(Route – non-transitional)**

Copiii se plimbă prin parc.  
children.DEF REFL.3PL stroll.3PL PRIN park

‘The children are strolling (around) through the park.’

(46) The ON series

a. **(Place)**

Creionul este pe masă.  
pencil.DEF is PE table

---

<sup>5</sup>Other series like ABOVE and BEHIND would involve (at least) another head, AxPart, which comes before Place, on top of the DP. To keep derivations relatively simple, I do not discuss these series here.

‘The pencil is on the table.’

b. **(Goal – transitional)**

Muncitorii s-au urcat pe clădire.  
workers.DEF REFL-have.3SG got up PE building

‘The workers got up on the building’

c. **(Source – transitional)**

Șoarecele a fugit de pe scaun.  
mouse.DEF have.3SG run DE PE chair

‘The mouse ran away from its position on the chair.’

d. **(Route – transitional)**

Copilul a sărit peste gard.  
child.DEF have.3SG jumped PESTE fence

‘The child jumped over the fence’

e. **(Route – non-transitional)**

Acrobatul merge pe o sfoară.  
acrobat.DEF walk.3SG PE a rope

‘The acrobat is walking on a rope.’

(47) The AT series

a. **(Place)**

Lucrez la magazin.  
work.1SG LA store

‘I work at the store.’

b. **(Goal – transitional)**

George merge la școală.  
George go.3SG LA school

‘George is going to school’

c. **(Source – transitional)**

George pleacă de la petrecere.  
George leave.3SG DE LA party

‘George is leaving from the party.’

d. **(Route – transitional)**

Trec pe la Dan întâi.  
pass.1SG PE LA Dan first

‘I will pass by Dan’s place first.’

e. **(Goal – non-transitional)**

Mergi (în)spre/către dreapta și îl găsești.  
go.2SG (ÎN)SPRE/CĂTRE right and CL.ACC find.2SG

‘Go to the right and you will find it’

f. **(Source – non-transitional)**

Vine dinspre est.  
come.3SG DINSPRE est

‘He/She/It is coming from the east.’

g. **(Route – non-transitional)**

Mergea pe la marginea drumului.  
walk.3SG.IMPF PE LA edge road.GEN

‘He/She/It was walking along the edge of the road’

(48) The UNDER series

a. **(Place)**

Valiza este sub pat.  
suitcase.DEF is SUB bed

‘The suitcase is under the bed.’

b. **(Goal – transitional)**

Pisica a fugit sub pat.  
cat.DEF have.3SG run SUB bed

‘The cat ran under the bed.’

c. **(Source – transitional)**

Pisica a fugit de sub pat.  
cat.DEF have.3SG run DE SUB bed

‘The cat ran from under the bed.’

d. **(Route – transitional)**

Am trecut cu mașina pe sub pod.  
have.1SG passed with car.DEF PE SUB bridge

‘I passed through under the bridge in the car.’

e. **(Route – non-transitional)**

Peștii înoată în cercuri pe sub apă.  
fish.DEF swim.3PL in circles PE SUB water

‘The fish are swimming in circles under water.’

If we systematize the directional expressions from the examples in (45-48), we get the following prepositions:

|                         | SERIES        | IN                       | ON    | AT                       | UNDER  |
|-------------------------|---------------|--------------------------|-------|--------------------------|--------|
|                         | <b>Place</b>  | în                       | pe    | la                       | sub    |
| <b>Transitional</b>     | <b>Goal</b>   | în                       | pe    | la                       | sub    |
|                         | <b>Source</b> | din < de + în            | de pe | de la                    | de sub |
|                         | <b>Route</b>  | prin < p(r)e + în        | peste | pe la                    | pe sub |
| <b>Non-transitional</b> | <b>Goal</b>   | (în)spre / către         | ?     | (în)spre / către         | ?      |
|                         | <b>Source</b> | dinspre < de + în + spre | ?     | dinspre < de + în + spre | ?      |
|                         | <b>Route</b>  | prin < p(r)e + în        | pe    | pe la                    | pe sub |

Table 1: Romanian directional expressions

With Table 1 in hand, we can now proceed to determining the shapes of the lexical items.



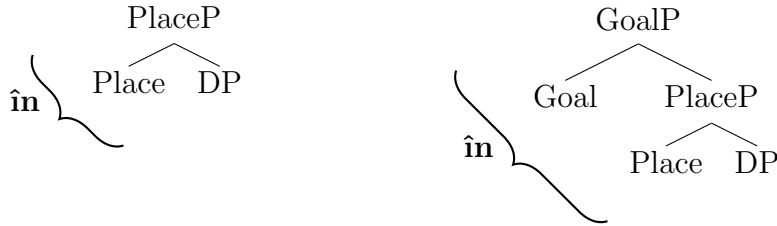
## 4.1 Spanning

According to the data, the different types of Path should be built as follows.

### The IN series

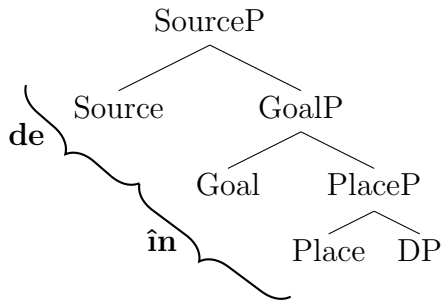
In the IN series, the preposition *în* is used for locative constructions and transitional Goal Paths. Thus, it appears that the preposition spans Place and Goal.

- (49) a. Location                      b. Transitional Goal Path



For a transitional Source Path, the morpheme *de* is added to the Goal Path. In this series, *de* + *în* is contracted into *din*.

- (50) Transitional Source Path



For a Route Path we add *pe* to a Goal expression. *pe* + *în* is contracted into *prin*<sup>6</sup>.

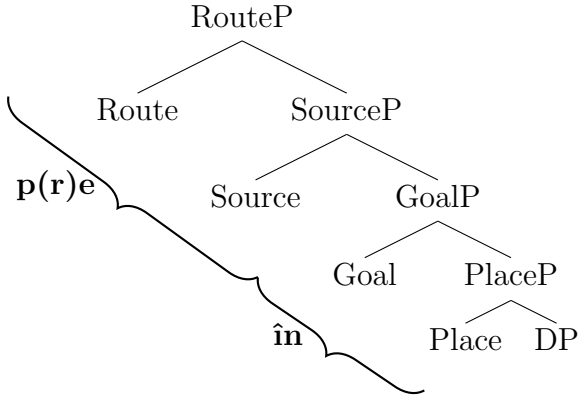
The reason why it must be the case that *pe* is added to a Goal expression directly and not to a Source one is that if *pe* was stacked on Source, *de* would appear as well, to lexicalize Source. We would get *\*pedein* instead of *prin*.

- (51) \*                      RouteP
- 

<sup>6</sup>*i* and *â* are Romanian spelling variants for the vowel /i/. The etymology of *din* and *prin* given here is further supported by their frequent pronunciation as [din] and [p(r)in] respectively.

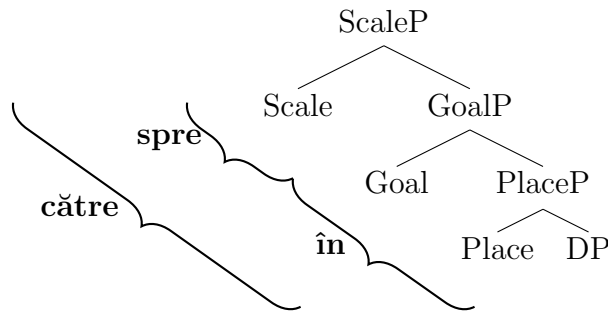
In order to get *prin*, *de* has to be overridden when the Route feature is added. Because of the ‘Biggest Wins’ principle, this would happen if there were a ‘bigger’ lexical item that contained what is already lexicalized by *de* as well, meaning the Source feature. Thus, the item *pe* spans over Route and Source.

(52) Transitional Route Path



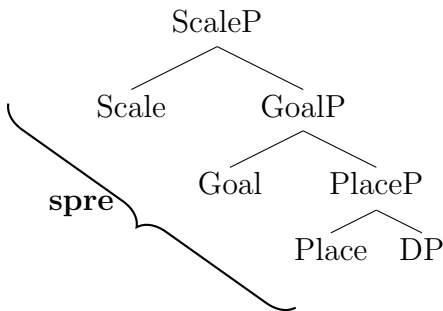
Going to non-transitional Paths, *spre*, *înspre* and *către*<sup>7</sup> can be used to express a Goal Path.

(53) Non-transitional Goal Path



The fact that *spre* alone can express a non-transitional Goal Path may be taken as an indication that actually *spre* contains Goal and Place as well. Indeed, if *spre* can stand alone, it cannot lexicalize only the Scale head, since Scale does not do anything by itself. If Scale is merged, we are dealing with at least a Goal Path in its complement.

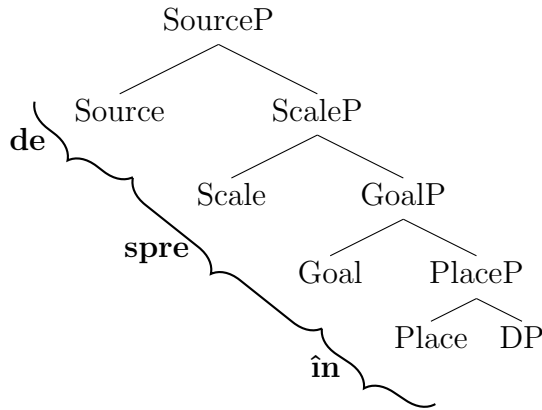
(54) Non-transitional Goal Path



<sup>7</sup>It would be interesting to see if there is any difference in interpretation between these three variants. In particular, it is strange that *spre* and *în + spre* can both be used to express the same Path. Even if *în*, which would lexicalize Place and Goal, is missing, *spre* can still stand alone, which may be taken as a clue that *spre* itself contains the Goal and Place heads, as shown in (54). But if this is the case, where does the optionality stem from? According to ‘Biggest Wins’, *spre* should override *în* and *în* should not show up at all.

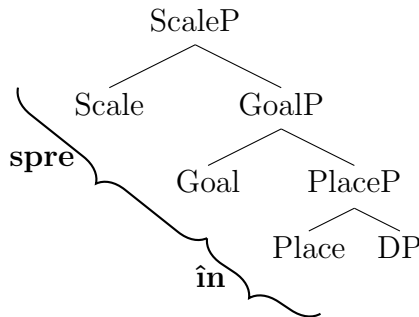
For a Source Path, we add *de* to the Goal Path. We see from Table 1 that, when composing a non-transitional Source Path, *în* from *în-spre* is obligatory. It must be *de-în-spre*, not *\*de-spre*<sup>8</sup>. This shows that *în* appears, to lexicalize (at least) Place. Taking this and (54) into consideration, it would not be unreasonable to restrict the span of *în* to Place, leaving *spre* to lexicalize Goal and Scale, while *de* will spell-out Source, just as it does in transitional Source Paths. Therefore, Romanian chooses option (39b) to build a non-transitional Source expression, by attaching Scale to a Goal Phrase (lexicalized by *spre*), and then a Source head on top.

(55) Non-transitional Source Path



According to the span of *spre* in (55), a non-transitional Goal Path with *în-spre* would look like in (56) below.

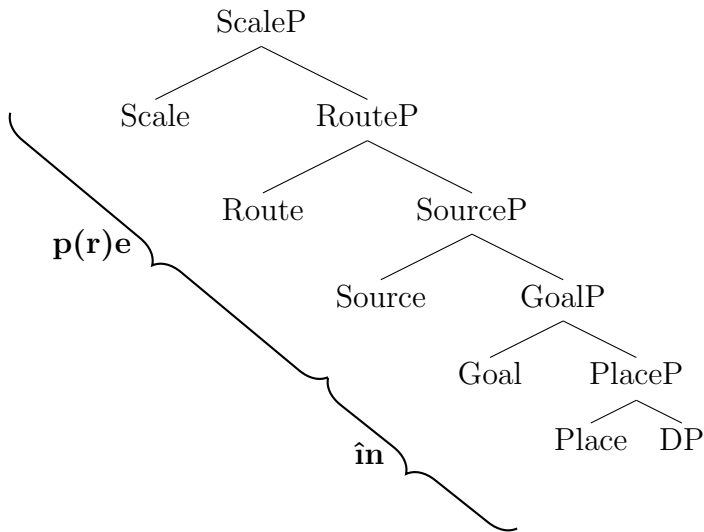
(56) Non-transitional Goal Path



The non-transitional Route Path is syncretic with its transitional counterpart, both expressed by *prin*. We have seen from transitional Path spans that *p(r)e* contains Route and Source. The syncretic non-transitional Route Path suggests adding the Scale head to the span of *p(r)e*.

<sup>8</sup>*despre* exists in Romanian, as a preposition meaning ‘about’. It is not used in directional expressions.

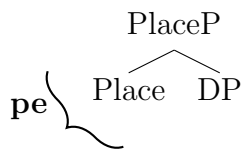
(57) Non-transitional Route Path



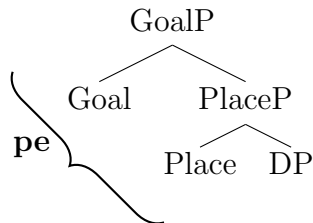
**The ON series**

As the IN series, ON has its own characteristic Place and Goal preposition, *pe*. *pe* seems to behave similarly to its counterpart *in*, in that it is used in both locative and transitional Goal Path constructions.

(58) a. Location

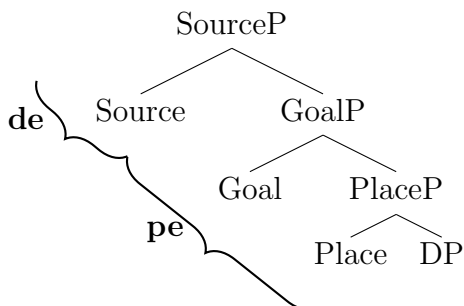


b. Transitional Goal Path



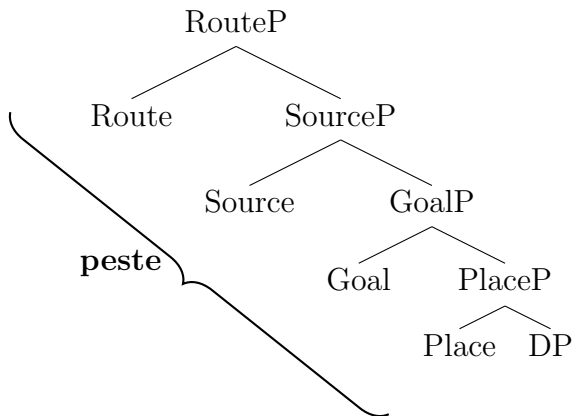
The transitional Source Path is built by adding the Source morpheme *de*.

(59) Transitional Source Path



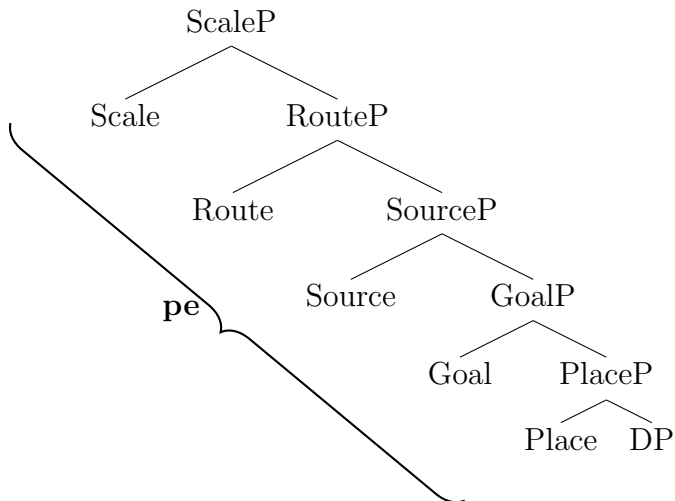
For the transitional Route Path we would expect *pe pe*. Instead, we get a single preposition, *peste*. This preposition spans the entire syntactic structure.

(60) Transitional Source Path



The non-transitional Route Path has one big *pe* morpheme spanning the entire structure.

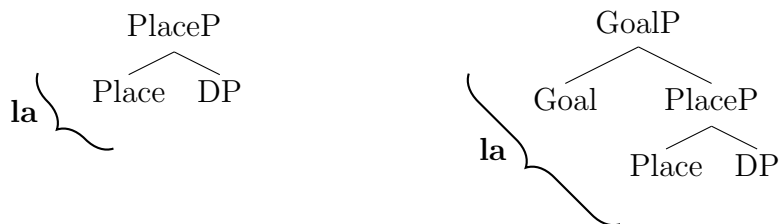
(61) Transitional Route Path



### The AT series

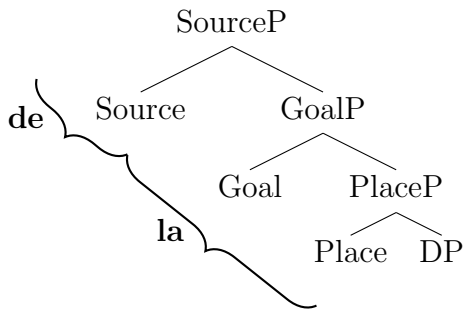
The preposition of the AT series is *la*, which spans Place and Goal.

(62) a. Location                      b. Transitional Goal Path



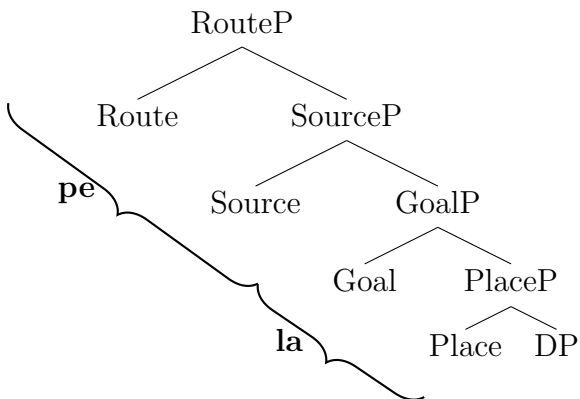
For the transitional Source Path, *de* is added.

(63) Transitional Source Path



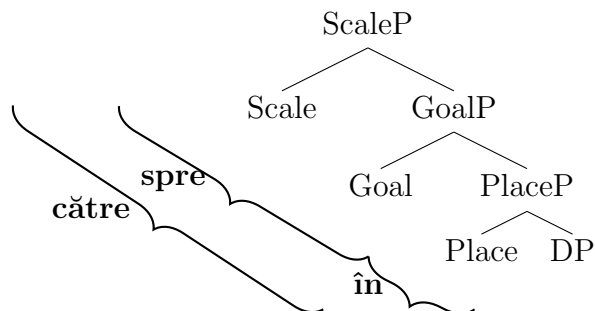
The expected *pe* appears in the transitional Route Path.

(64) Transitional Route Path

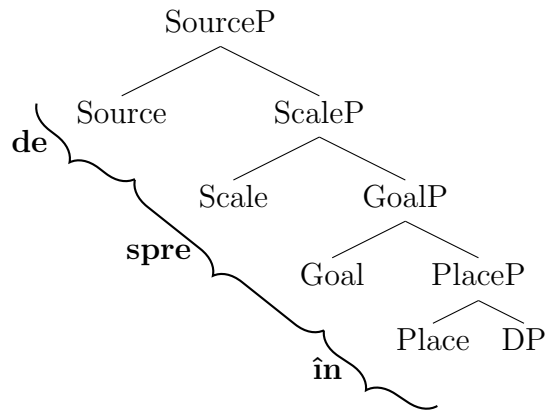


For non-transitional Goal and Source Paths, the AT series uses the same prepositions as the IN series, breaking the constructions with *la*.

(65) Non-transitional Goal Path



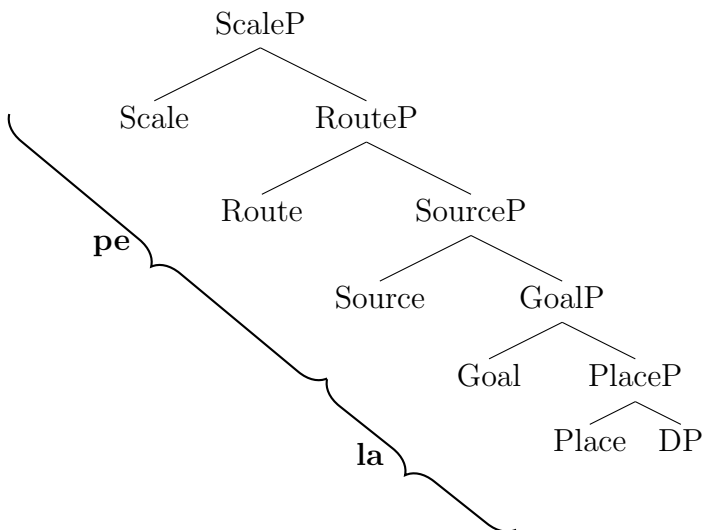
(66) Non-transitional Source Path



This appears to suggest that *în* and *spre* may be listed for two conceptual contents.

The non-transitional Route Path is built as expected, with the series preposition *la* and the Scale, Route, and Source *pe*.

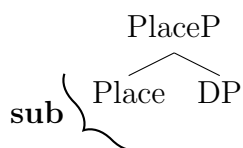
(67) Non-transitional Route Path



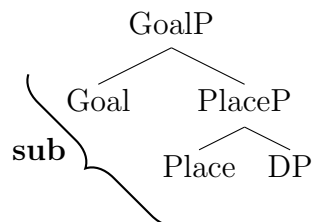
**The UNDER series**

Finally, the UNDER series, with its series marker *sub*, has Paths built in a predictable way.

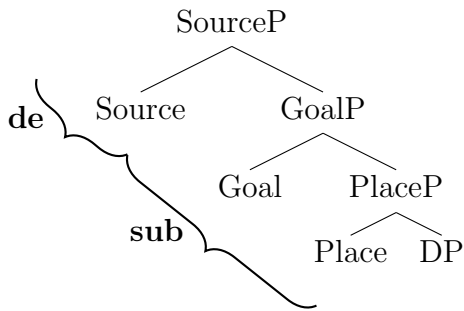
(68) a. Location



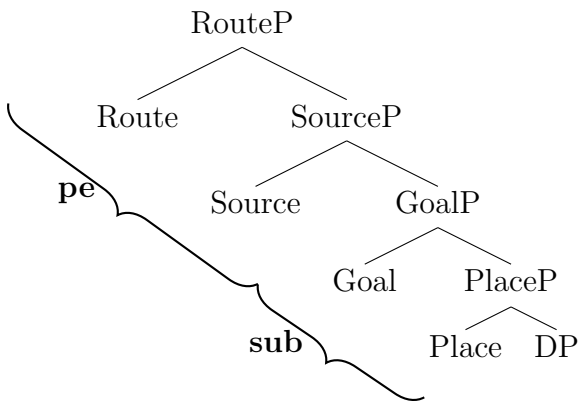
b. Transitional Goal Path



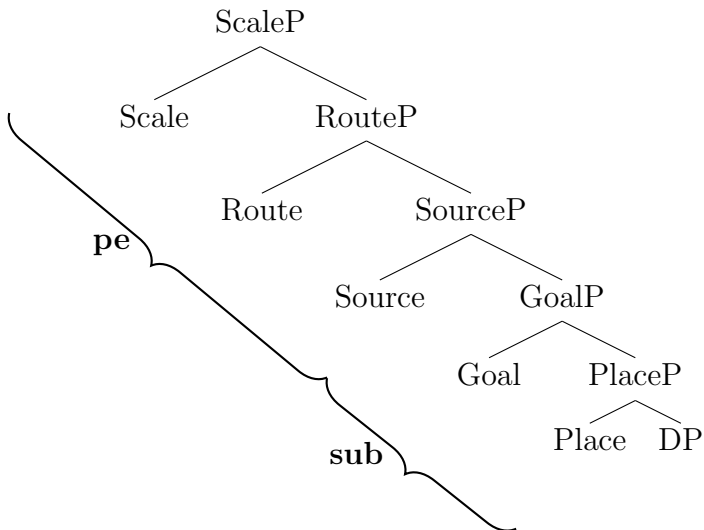
(69) Transitional Source Path



(70) Transitional Route Path



(71) Non-transitional Route Path



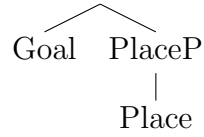
## 4.2 Lexical items with no phrasal nodes

Given the spans described in the previous subsection, we would expect the lexical items to look as shown below.

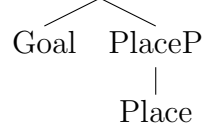


(72) Series markers

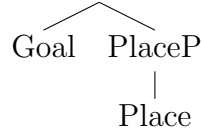
a.  $\hat{in} \Leftrightarrow </\text{in}/ , \text{GoalP} , \text{IN}, \text{AT} >$



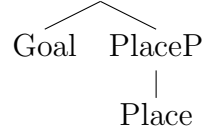
b.  $pe_1 \Leftrightarrow </\text{pe}/ , \text{GoalP} , \text{ON} >$



c.  $la \Leftrightarrow </\text{la}/ , \text{GoalP} , \text{AT} >$

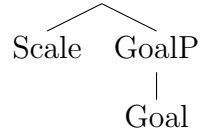


d.  $sub \Leftrightarrow </\text{sub}/ , \text{GoalP} , \text{UNDER} >$

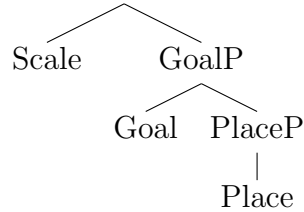


(73) Other morphemes with conceptual content

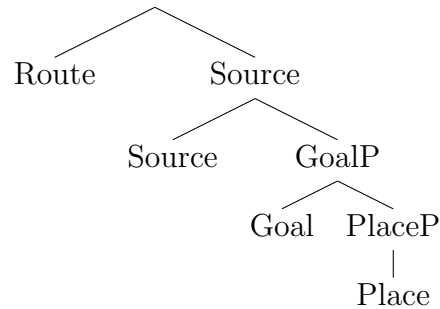
a.  $spre \Leftrightarrow </\text{spre}/ , \text{ScaleP} , \text{IN}, \text{AT} >$



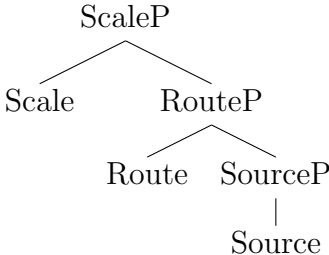
b.  $c\hat{a}tre \Leftrightarrow </\text{k}\hat{a}tre/ , \text{ScaleP} , \text{IN}, \text{AT} >$



c.  $peste \Leftrightarrow </\text{peste}/ , \text{RouteP} , \text{ON} >$

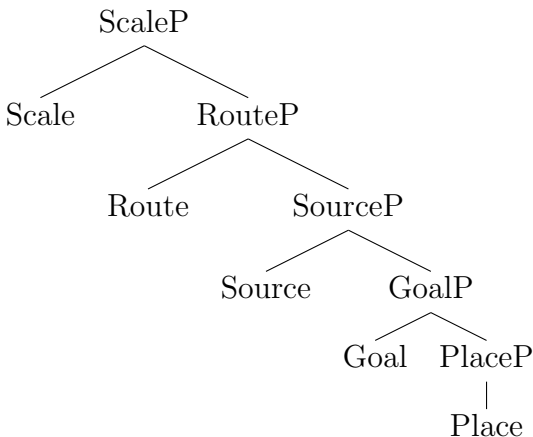


(74) Morphemes with no conceptual content

- a.  $de \Leftrightarrow </de/$  ,      Source       $>$
- b.  $pe_2 \Leftrightarrow </pe/$  ,        $>$

The reason why we need two *pe* morphemes is that one of them is a series marker associated with the conceptual content ON and lexicalizes only Goal and Place, while the other is common to all series, thus lacking conceptual content of its own, and lexicalizes Scale, Route, and Source. In terms of spanning, they appear to complement each other, rather than intersect, so there is no reason to consider that they might be the same lexical item. The homophony here may be accidental.

Notice that there is one lexical item missing from the list in (72-74), namely the ‘extra-large’ *pe* used for the non-transitional Route Path of the ON series. Conceivably, this could be another *pe*<sup>9</sup>.

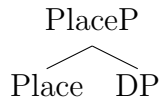
- (75)  $pe_3 \Leftrightarrow </pe/$  ,       , ON  $>$

Notice that, as with the transitional ON Route Path, we would expect it to be expressed with *pe pe*. One *pe* is *pe*<sub>1</sub> from the series marker stretching over Place and Goal, and the other *pe* would be *pe*<sub>2</sub>, lexicalizing the rest of the structure, up to Scale. For the transitional Route Path, there is the tailor-made item *peste*, overriding the analytical expression *pe*<sub>1</sub> + *pe*<sub>2</sub>. Since the even bigger non-transitional Route Path is expressed as *pe*, we must consider a third *pe*, which contains the entire structure and overrides *peste* with the addition of Scale.

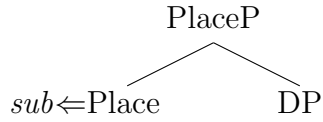
Going back to the details of the lexically specified trees, there is one more issue that needs to be resolved before deciding on the shapes of the directional prepositions in Romanian. In order to identify the problem, let us proceed with a derivation. The derivation steps for building UNDER series Paths are illustrated in what follows.

<sup>9</sup>This situation is potentially problematic, since it appears to be a violation of the \*ABA generalization, according to which syncretism targets only adjacent heads, such that the syncretism of A and C to the exclusion of B is prohibited (see Pantcheva (2011: 223-226) for a discussion). In this paper, I abstract away from this issue.

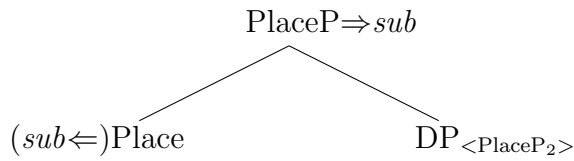
- (76) a. Step 1: merge Place with DP



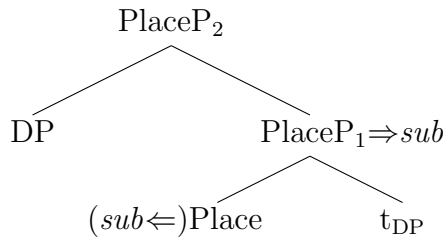
- b. Step 2: insert *sub* at node Place



- c. Step 3: insert *sub* at node PlaceP; mark DP for extraction



- d. Step 4: evacuate DP

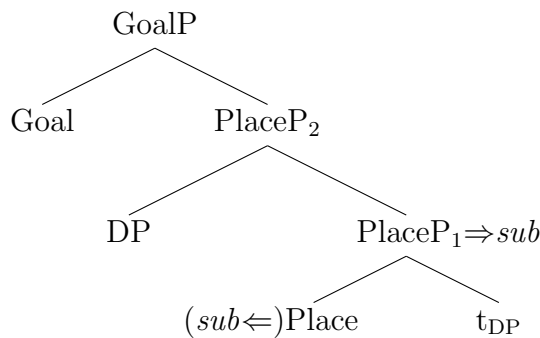


The result is a postposition instead of a preposition.

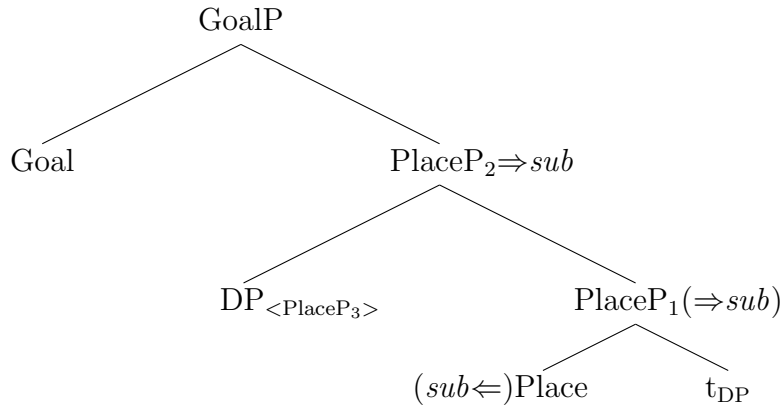
- (77) a. \*DP sub  
         DP under  
       b. \*masă sub  
           table under  
           ‘under the table’  
       c. sub masă  
           under table  
           ‘under the table’

Let us now move on to the Goal Path.

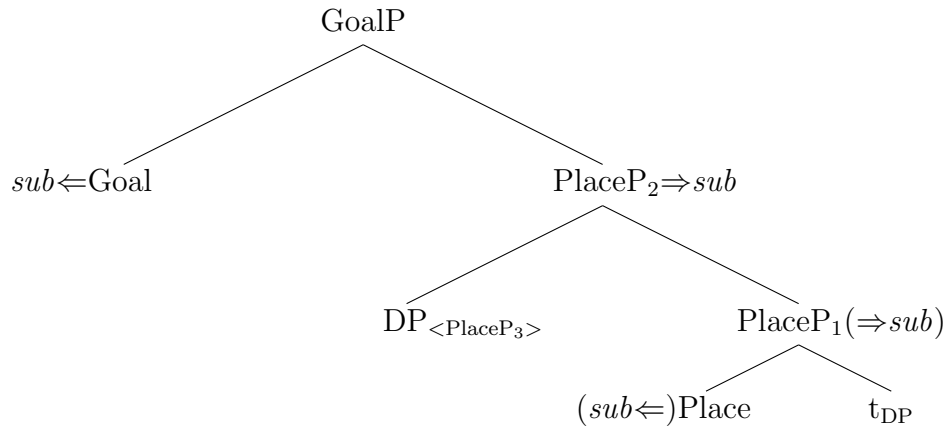
- (78) a. Step 1: Merge the Goal head



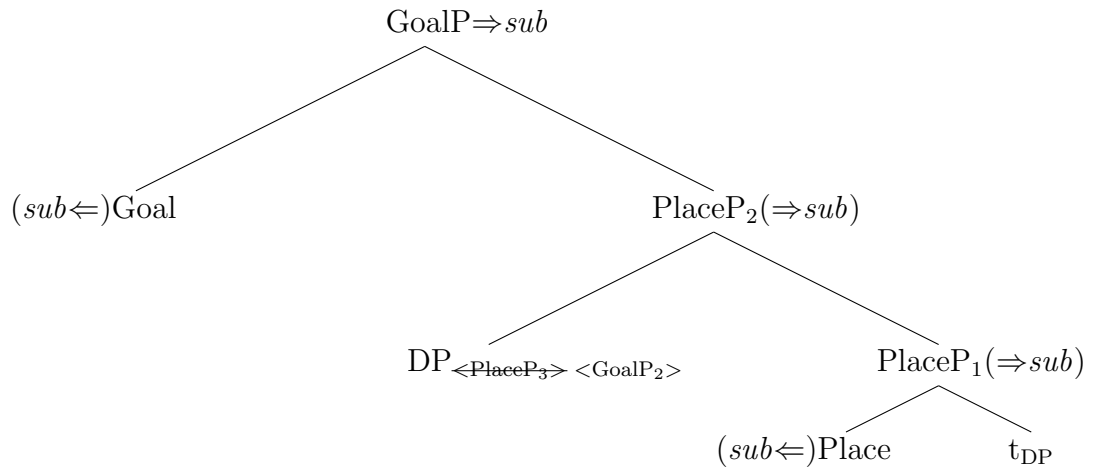
- b. Step 2: inspect PlaceP<sub>2</sub> for lexicalization; insert *sub* and mark DP for extraction



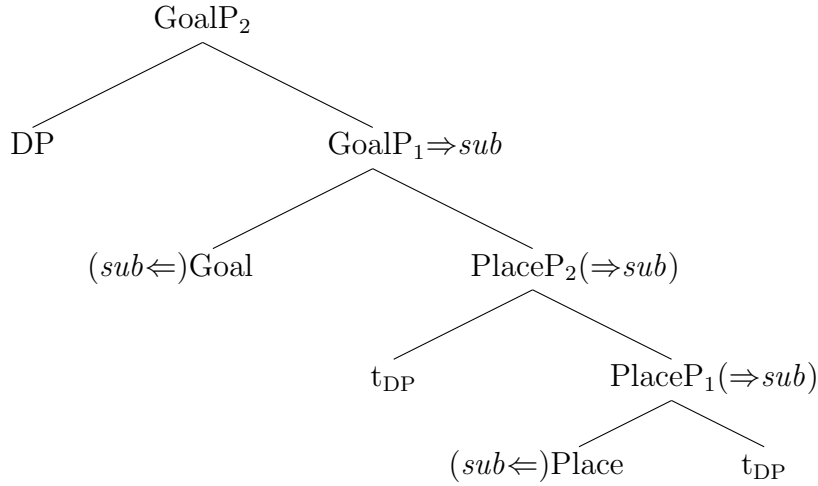
- c. Step 3: insert *sub* at Goal



- d. Step 4: insert *sub* at GoalP and mark DP for extraction to GoalP<sub>2</sub>



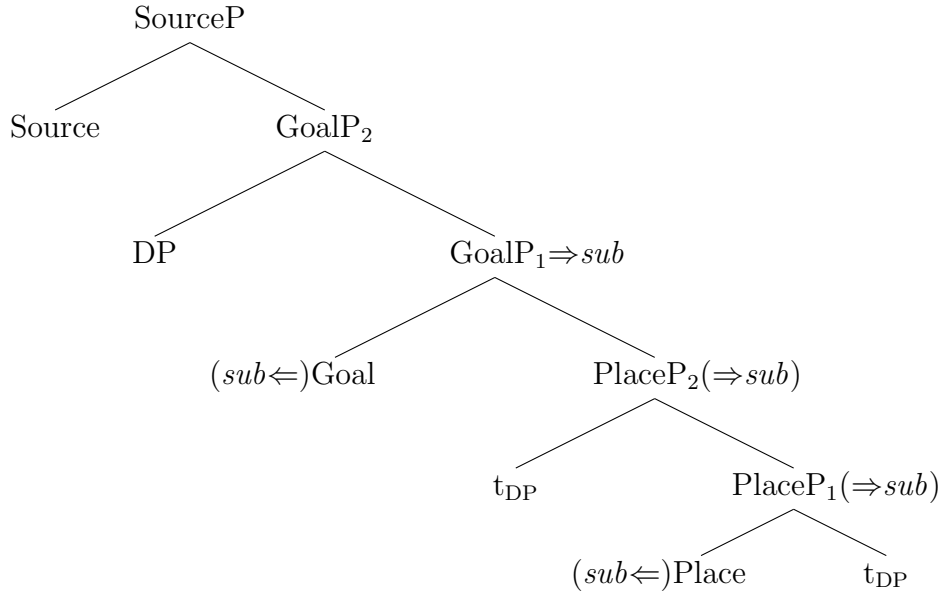
e. Step 5: evacuate DP



The effect seen in the locative construction takes place again. The DP ends up above, and will be pronounced before, what should be a preposition.

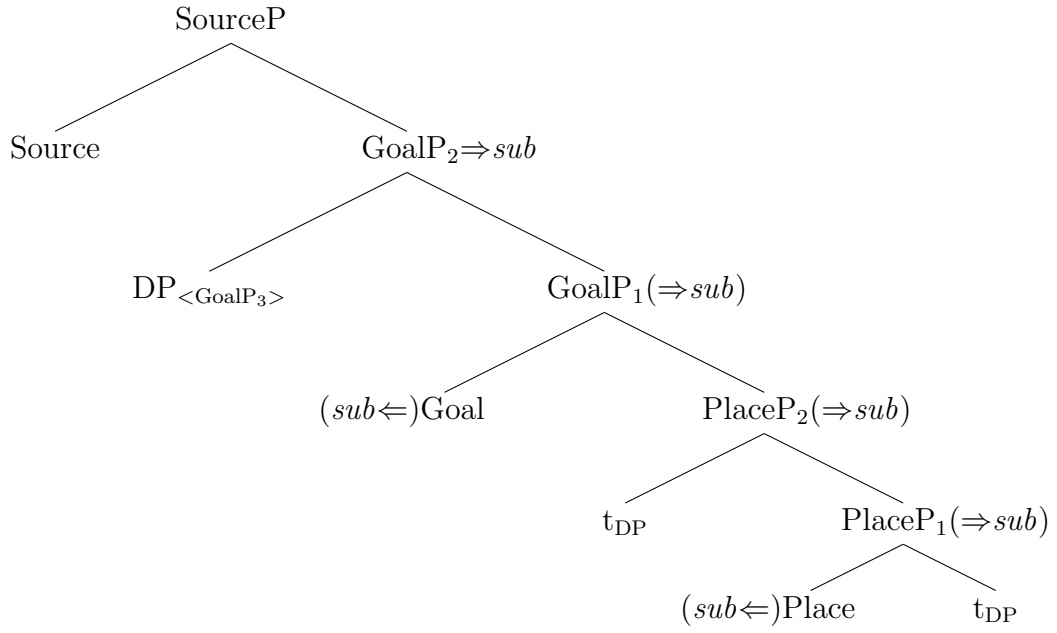
Let us see what happens if we add Source to the equation.

(79) a. Step 1: Merge Source

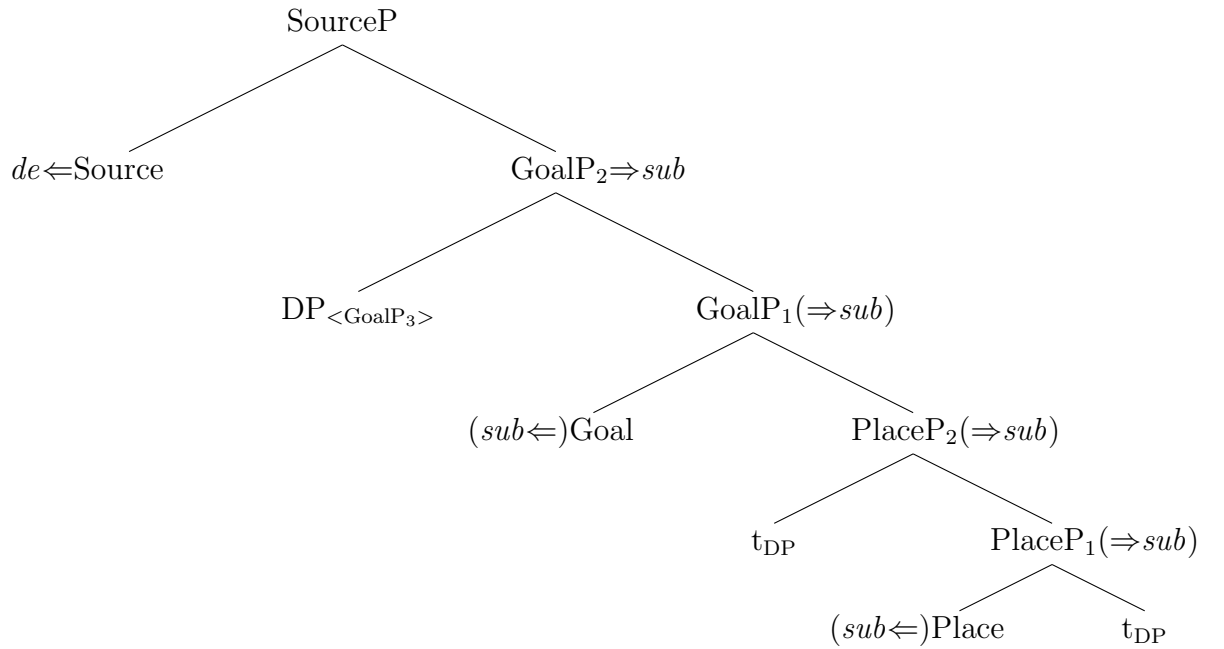


b. Step 2: inspect GoalP<sub>2</sub> for lexicalization; insert *sub* and mark DP for extraction to GoalP<sub>3</sub><sup>10</sup>

<sup>10</sup>Another possible landing site is SourceP<sub>2</sub>, but the system described in Pantcheva (2011) goes with the shortest move. In this case, DP just has to move from under the domain of GoalP<sub>2</sub>, so the shortest move lands it immediately above GoalP<sub>2</sub>, right before Source.



c. Step 3: inspect Source; insert *de*



d. Step 4: inspect SourceP; there is no lexical item containing that node; the derivation is convergent because SourceP is lexicalized by inheritance

In whis case, the DP lands between two prepositions even after having moved to GoalP<sub>3</sub>.

- (80) a. *de* DP sub  
       from DP under  
       b. *de* masă sub  
           from table under  
           'from under the table'

Consider how *de* and *sub* differ in their lexical specification. *sub* has phrasal nodes, while *de* does not have them. Let us look at what would happen if *de* did contain the phrasal node of

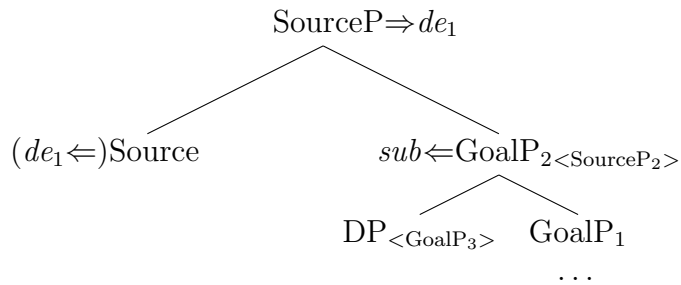
its only feature.

$$(81) \quad de_1 \Leftrightarrow \langle /de/ , \text{SourceP} \rangle$$

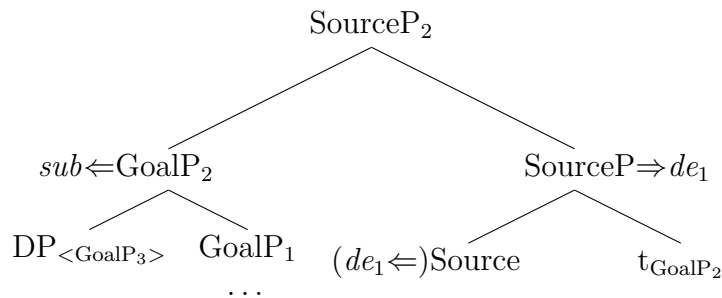
$\downarrow$   
 Source

At Step 4,  $de_1$  would be found to be a matching node for SourceP and would be inserted. This is illustrated below, redoing Step 4.

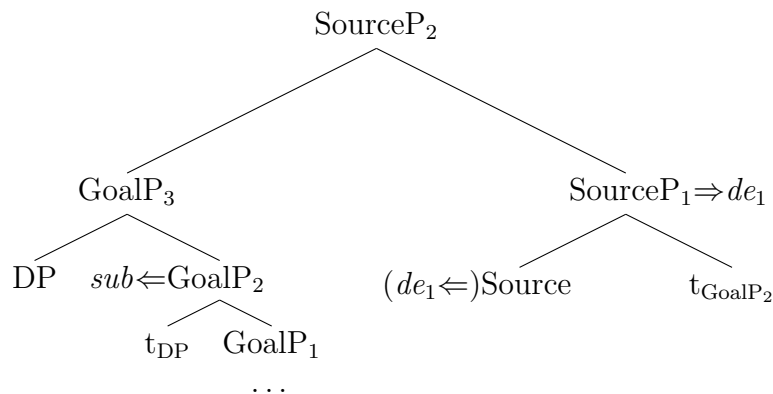
- (82) a. Step 4: inspect SourceP; insert  $de_1$  and mark GoalP<sub>2</sub> for extraction to SourceP<sub>2</sub>



- b. Step 5: move GoalP<sub>2</sub> to SourceP<sub>2</sub>



- c. Step 6: move DP to GoalP<sub>3</sub><sup>11</sup>



At the end of this rather complicated derivation, we end up with the DP preceeding the Place/Goal preposition, which in turn preceeds the Source preposition.

<sup>11</sup>Movements are performed in an order reverse to that in which they were triggered. Two moves were scheduled in the cycle in which Source was merged: DP to GoalP<sub>3</sub>, triggered by the lexicalization of GoalP<sub>2</sub>, and GoalP<sub>2</sub> to SourceP<sub>2</sub>, triggered by the insertion of *de*<sub>1</sub> at SourceP. The last scheduled movement occurs first, and the previous one occurs second.

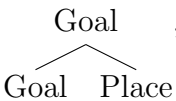
- (83) a. \*DP sub de  
 DP under from  
 b. \*masă sub de  
 table under from  
 ‘from under the table’  
 c. de sub masă  
 from under table  
 ‘from under the table’

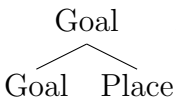
Notice that the movement was triggered when a lexical item was inserted at a phrasal node. *de* leaves GoalP in its place, while *de*<sub>1</sub> requires it to move. If the item is uttered after the DP, it means that the item has at least one phrasal node.

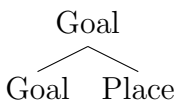
Notice also that *de* spans only one feature, while *sub* spans two. As lexical items have been presented up to this point, an item spanning more than one feature will automatically have phrasal nodes. How, then, do we get prepositions spanning more than one feature?

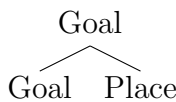
A solution for this problem is proposed in Caha (2010). He suggests that prepositional languages have lexical items that contain no phrasal nodes. In light of this proposal, the lexical items in (72-75) would look as follows.

- (84) Series markers

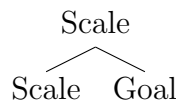
- a. *în* ⇔ </in/ ,  , IN, AT >

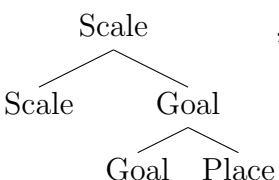
- b. *pe*<sub>1</sub> ⇔ </pe/ ,  , ON >

- c. *la* ⇔ </la/ ,  , AT >

- d. *sub* ⇔ </sub/ ,  , UNDER >

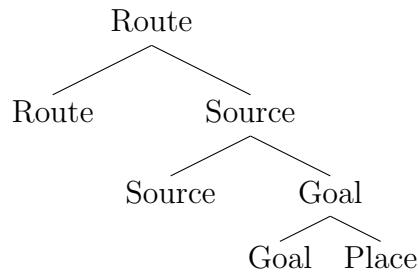
- (85) Other morphemes with conceptual content

- a. *spre* ⇔ </spre/ ,  , IN, AT >

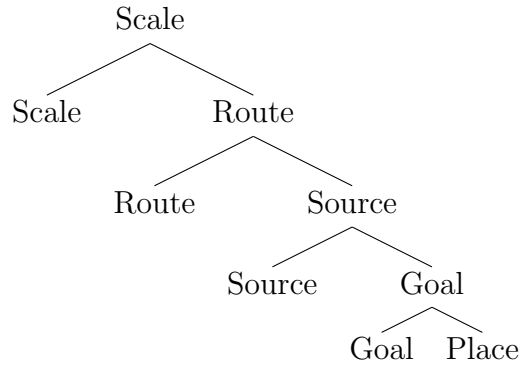
- b. *către* ⇔ </kətre/ ,  , AT >



c.  $peste \Leftrightarrow </peste/ , \quad , ON >$



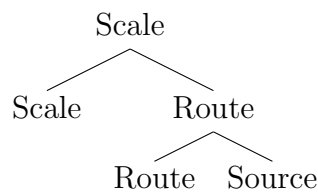
d.  $pe_3 \Leftrightarrow </pe/ , \quad , ON >$



(86) Morphemes with no conceptual content

a.  $de \Leftrightarrow </de/ , \quad \text{Source} \quad >$

b.  $pe_2 \Leftrightarrow </pe/ , \quad \text{Scale} \quad >$



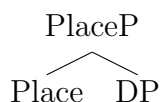
### 4.3 Deriving Romanian Paths

Armed with the lexical items above, we can now proceed to deriving different types of Path. In the remainder of this section, I will illustrate derivations so as to capture each ‘well-behaved’ Path type rather than go by series, since in many cases the same Path is built the same throughout series. Therefore, each Path will be exemplified with one series, after which I investigate Paths which employ tailor-made lexical items, or which have more problematic derivations, in the next section.

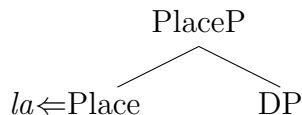
Let us, then, start with locative constructions, which are similar throughout series, modulo the different (phonological content of) series marking prepositions. I use the AT series to exemplify locative constructions in general.

(87) Locative construction, the AT series

a. Step 1: merge Place and DP



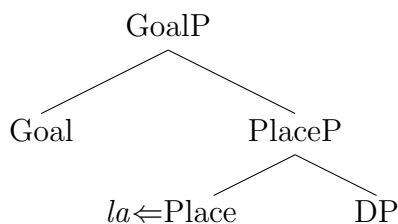
- b. Step 2: inspect node Place; insert *la*



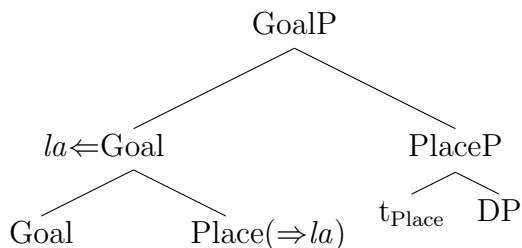
- c. Step 3: inspect PlaceP; there is no lexical item containing that node; the derivation is successful because PlaceP is lexicalized by inheritance

(88) Transitional Goal Path, the AT series

- a. Step 1: merge Goal



- b. Step 2: inspect Goal; *spre*, *către*, and *la* match; *la* is chosen because it contains no extra features; Goal is the topmost feature in *la*, so it cannot be inserted at Goal, but its exact lexical configuration can be reached by moving Place to adjoin to Goal



- c. Step 3: Inspect Goal P; there is no lexical item containing that node; the derivation converges, since GoalP is lexicalized by inheritance

In derivations with lexical items without phrasal nodes there will still be movement, but this movement is similar to head-to-head movement. It targets terminal nodes and leaves phrasal nodes out of the process. Because of this, there will not be movement of entire phrases, bringing the DP on top of the prepositions. Hence, prepositions remain prepositions instead of ending up as postpositions<sup>12</sup>.

Phrasal nodes will still appear when building the syntactic structure of the utterance, but no lexical item will be inserted, since none of the entries contains phrasal nodes. The derivation will still converge, since each phrasal node will be lexicalized by inheritance.

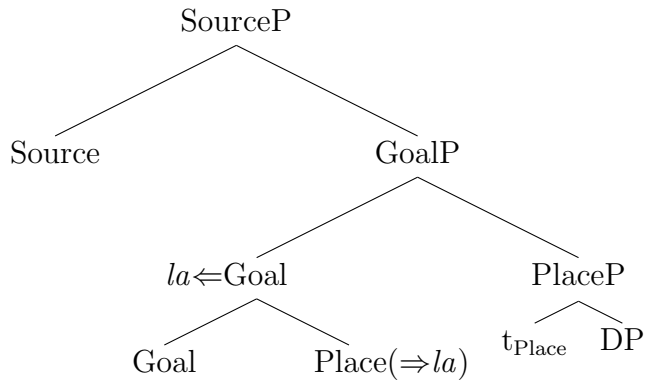
Proceeding to a Source Path and keeping with the AT series, we next add the Source head to the structure.

(89) Transitional Source Path, the AT series

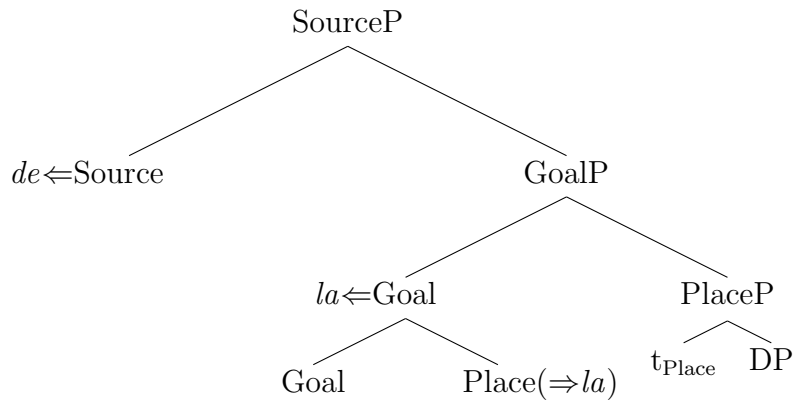
- a. Step 1: merge Source

---

<sup>12</sup>Or prefixes remain prefixes instead of turning into suffixes by moving the DP in front of them. The difference between prefixes and prepositions, and suffixes and postpositions is not relevant to the discussion. Indeed, if all are lexical items, this difference may be a non-issue altogether.



- b. Step 2: inspect Source; *de* and *pe*<sub>2</sub> match; choose *de* because it has no superfluous features; insert *de*

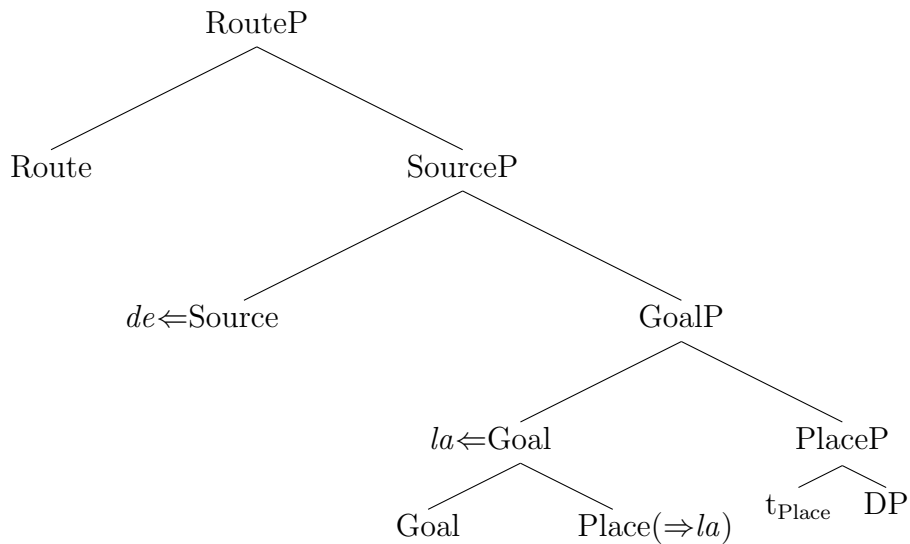


- c. Step 3: inspect SourceP; no item is inserted; SourceP is lexicalized by inheritance

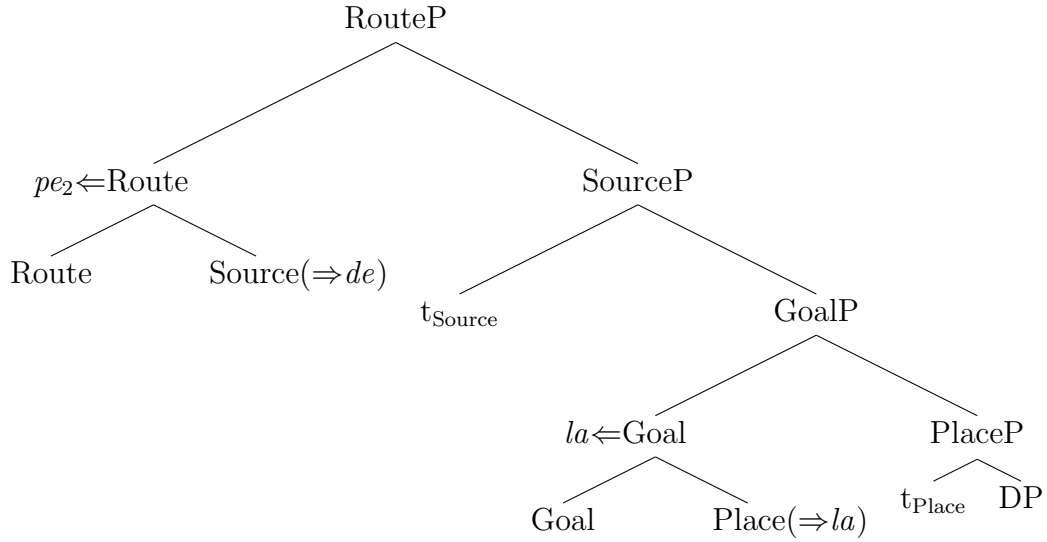
For the Route Path, the Route head is added on top of the Source expression.

(90) Transitional Route Path, the AT series

- a. Step 1: merge Route



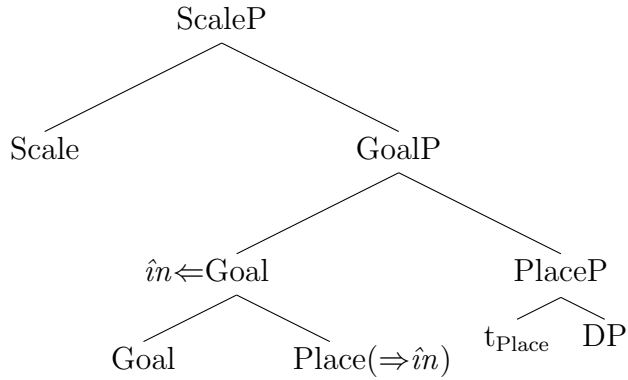
- b. Step 2: inspect Route;  $pe_2$  contains the Route feature, but it is the middle layer; the item can be inserted if Source adjoins to Route, creating a subconstituent of the specification of  $pe_2$



Let us now proceed to non-transitional Paths and switch to the IN series<sup>13</sup>. The Goal Path of the IN series is constructed by adding the Scale head to the transitional Goal Path.

(91) Non-transitional Goal Path, the IN series

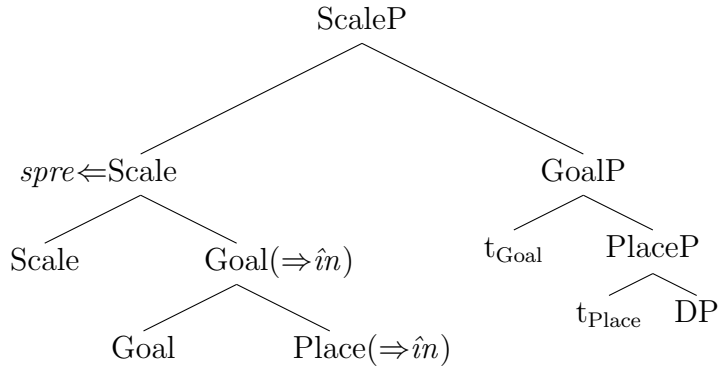
- a. Step 1: add the Scale head to the transitional Goal Path



- b. Step 2: inspect Scale; the matching items are *către* and *spre*; inserting *către* or *spre*<sup>14</sup> requires Goal (and Place) to adjoin to Scale

<sup>13</sup>For the moment, I leave aside the AT series, since there is a switch in prepositions in this series, making AT borrow from the IN series. Recall that the non-transitional Goal and Source Paths contain *în* rather than the series' own marker *la*.

<sup>14</sup>For the purposes of this derivation I use a variant of *spre* that contains Scale, Goal and Place. I leave the discussion of how to better capture the lexical entry of *spre* for the next section.



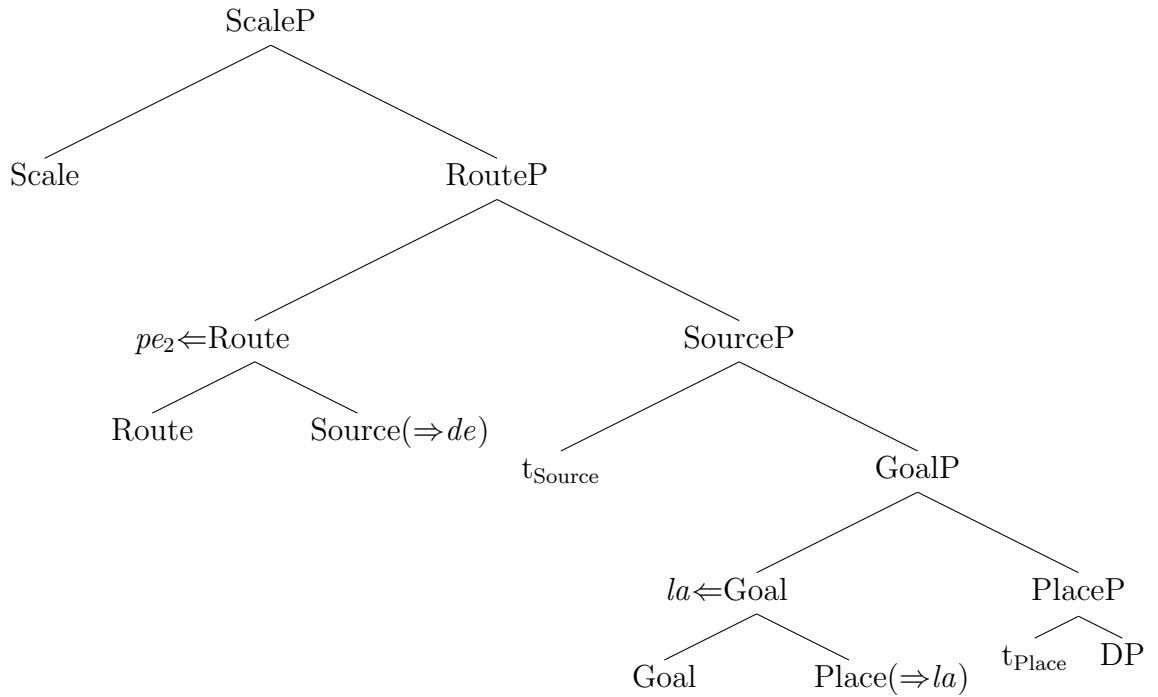
- c. Step 3: inspect ScaleP; there are no items and the node is lexicalized by inheritance

Notice that, when Scale calls for Goal to adjoin to it, Goal and Place already form an item, and it is this item that moves, creating the exact tree configuration of *către* or *spre*.

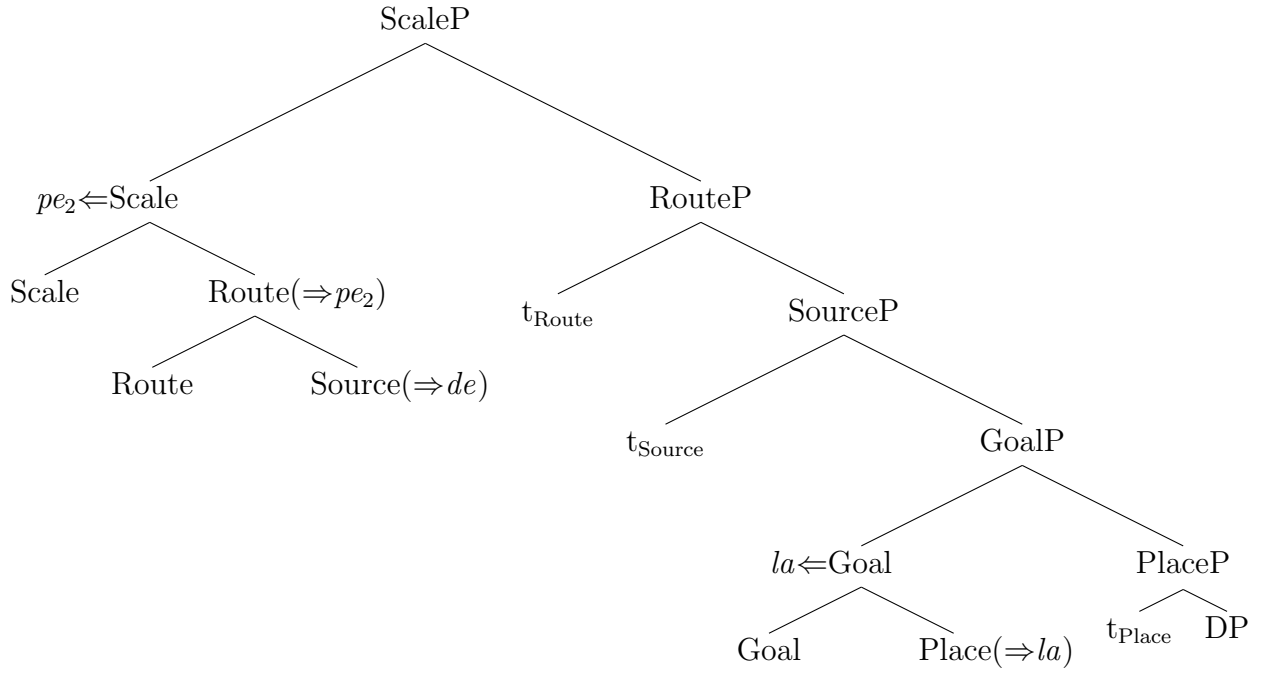
Leaving the discussion of non-transitional Source Paths for the next section of the paper, we go to the non-transitional Route Paths, going back to the AT series.

(92) Non-transitional Route Path, AT series

- a. merge Scale to the transitional Route Path



- b. inspect Scale;  $pe_2$  matches, but inserting this item requires Route to adjoin to Scale, in order to create the right syntactic configuration



c. Step 3: inspect ScaleP; no item is inserted and the node is lexicalized by inheritance

## 5 Thorny issues

While the previous section out-lined the derivations of the well-behaved expressions of Table 1, this section delves into some of the more difficult. The questions tackled concern the interchangeability of *spre* and *inspre*<sup>15</sup>, and the tailor-made entry *peste* of the ON series.

### 5.1 *spre* and *inspre*

This subsection works towards establishing the lexical entry of *spre*, in order to illustrate the derivation of non-transitional Goal and Source Paths for the IN and AT series. The main questions are the following. How come *spre* and *inspre* can be used interchangeably? Also, recall that the Source Path expression is *de* + *inspre*, never *de* + *spre*. Why is it that the non-transitional Source Path requires *inspre* if the two expressions are equivalent?

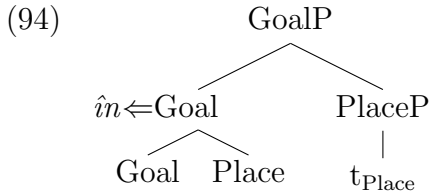
In the previous section I argued that *spre* must lexicalize Scale and at least Goal, given that it can express a non-transitional Goal Path by itself. What is more, in the case that *spre* is the only lexical item employed in the lexicalization of the non-transitional Goal Path, it should lexicalize Place as well. By this line of reasoning, the features of *spre* are Scale, Goal, and Place. Therefore, it should have the lexical specification in (93):

$$(93) \quad spre \Leftrightarrow \langle /spre/ , \quad \begin{array}{c} \text{Scale} \\ \swarrow \quad \searrow \\ \text{Scale} \quad \text{Goal} \\ \quad \swarrow \quad \searrow \\ \quad \text{Goal} \quad \text{Place} \end{array} , \text{IN, ON} \rangle$$

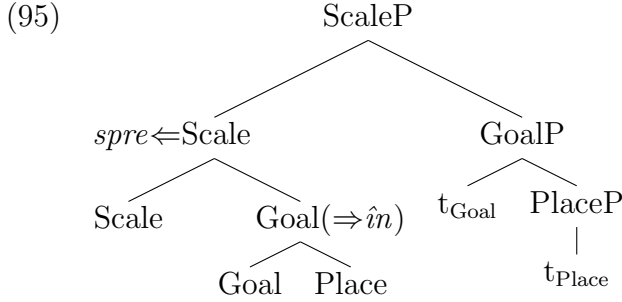
Recall that *în* is the preposition that contains Goal and Place. Thus, Place, and possibly Goal as well, are sometimes lexicalized by a more specific preposition, in cases of *inspre*.

The first problem with the interchangeability of *spre* and *inspre* is that this situation is a violation of the ‘Biggest wins’ principle. When Place is merged, *în* is inserted. Goal is next, and *în* is a match again.

<sup>15</sup>I leave *către* aside for this discussion, since it contains the structure of the entire Path, posing no problems.

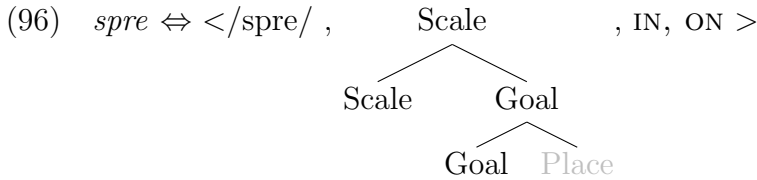


When Scale is merged, the only suitable item is *spre*. The configuration for *spre* is created, and *spre* overrides *in*.



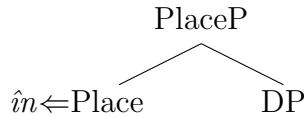
If *in* is overridden by a bigger lexical item, how come it can optionally still appear? Another problem is the prepositions' relative ordering. Suppose *spre* did contain only Scale and *in* spelled-out Goal and Place. What pushes *in* in front of *spre*? *in* should stay in its place, with *spre* lexicalizing Scale above it, and the entire structure should be spelled-out as *sprein*.

A possible solution is to consider that Place (and possibly Goal as well) are 'fading in and out' of entry *spre*'s specification<sup>16</sup>. Let us consider a *spre* entry with a 'flicking light bulb' Place head<sup>17</sup>:



When the Place light bulb is flicked on, *spre* can cover the entire structure by itself. The derivation looks like in (95) (or (91b)), with *in* overridden. When Place is off in the entry of *spre*, *in* has to step in and fill the gap. This is why we end up with *in* surfacing.

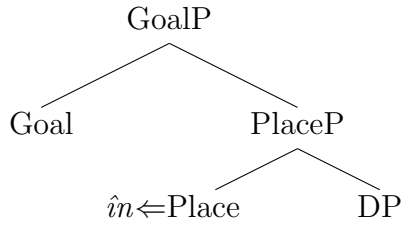
- (97) a. Step 1: inspect Place; insert *in*



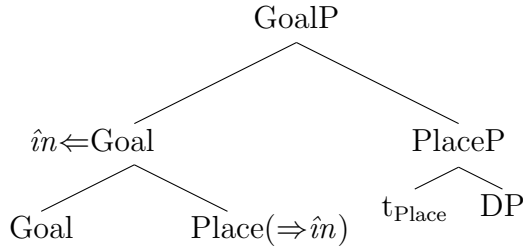
- b. Step 2: add Goal

<sup>16</sup>Presumably, we might generalize this line of thought as an explanation for microvariation or diachronic change. Suppose a feature light bulb flicks for a period of time and then shuts down for good. Lexical entries may lose features from their specifications, which would lead to a different course of the derivation. In this case, a question arises: can we find rules according to which entries may lose their features? For example, it might be expected that they lose either layers from the top or from the bottom of the tree, but not from the middle.

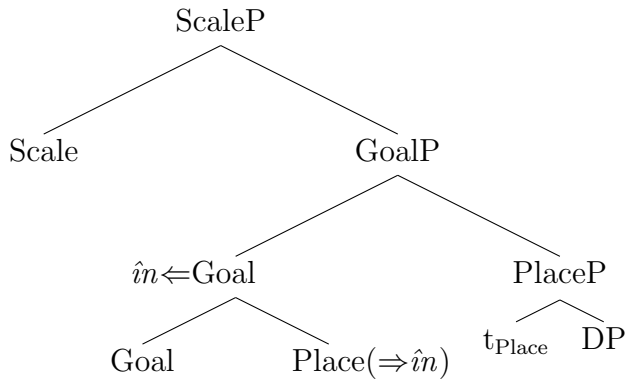
<sup>17</sup>I will take *spre* to contain Goal at all times because of its strong Goal-oriented directional connotation, even when uttered alone, out of context.



- c. Step 3: inspect Goal; insert  $\hat{in}$  and adjoin Place to Goal, creating the exact configuration of  $\hat{in}$



- d. Step 4: add Scale



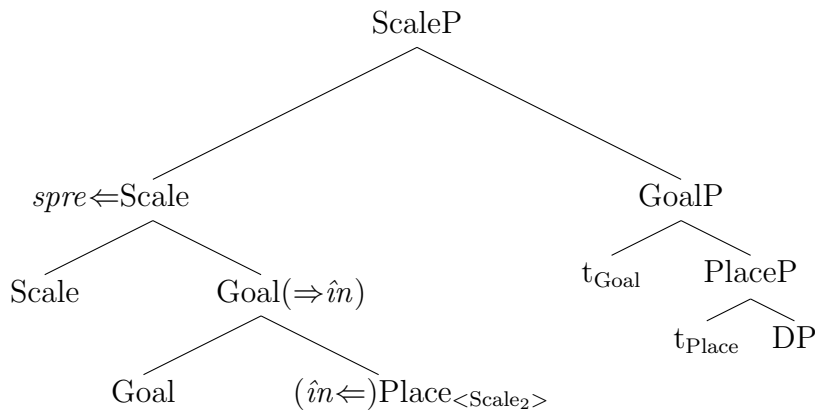
At this step, Scale needs lexicalization. The only item that can spell it out is *spre*, with its Place feature bulb turned off (which reduces it to the specification in (98) below).

$$(98) \quad spre \Leftrightarrow </spre/ , \quad \begin{array}{c} \text{Scale} \\ \swarrow \quad \searrow \\ \text{Scale} \quad \text{Goal} \end{array} , \text{IN, ON} >$$

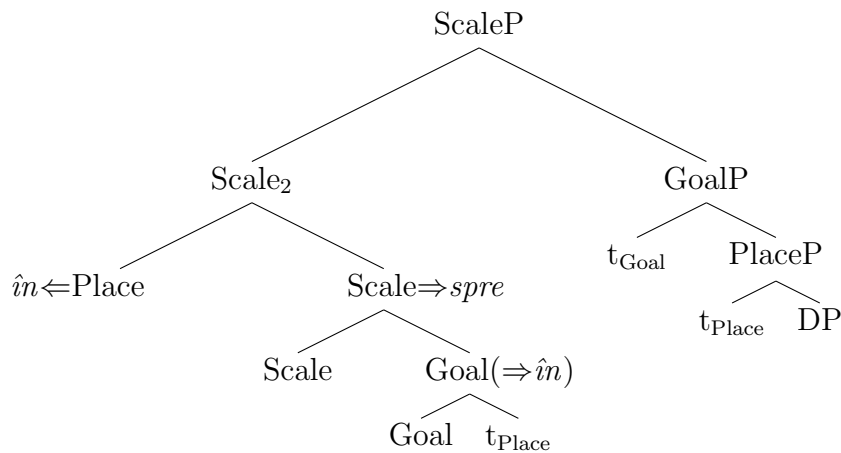
However, *spre* cannot be employed by making use of its topmost layer, Scale, while disregarding Goal. If Goal is adjoined to Scale in order to create the configuration of *spre*, Place moves along with it. In that case, Place would be in the way, since the entry of *spre* does not contain it. The solution in this case would be movement. *spre* can spell-out Scale (and Goal), if Place moves above Scale.

- (99) a. Step 5: insert *spre* at Scale and adjoin Goal to Scale





b. Step 6: move Place to Scale<sub>2</sub>



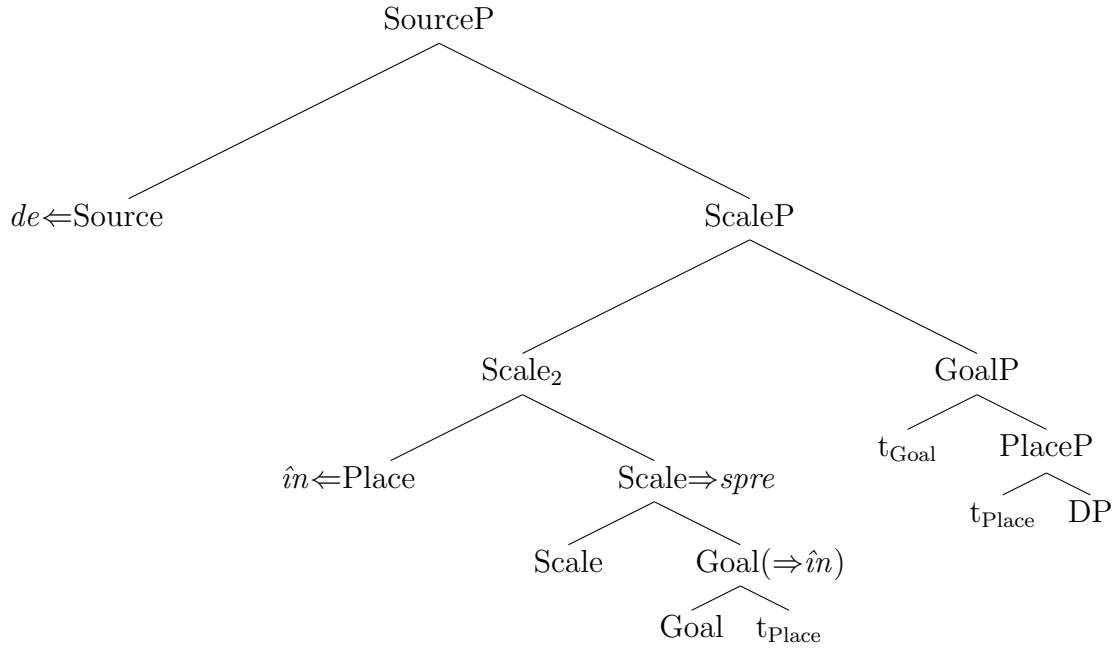
Creating the configuration of *spre* requires two movements: adjoining needed material (Goal) and evacuating surplus material (Place). The situation brings to attention the question of when these movements take place and in what order. The assumption we have been working under is that movements take place at the beginning of the cycle following the one in which they were triggered, in an order reverse to the one in which they were triggered. Thus, an item can be inserted at a node, with the promise that missing material will be adjoined in the following cycle. As far as I can see, there is no reason to assume that, when moving terminal nodes, additions must take place before inserting the lexical item, since the insertion triggered the movement in the first place.

In (99), the Goal feature is needed, but it comes carrying Place with it, and Place must be evacuated. Thus, the reasonable assumption is that the ‘adjoin Goal’ movement is triggered before the ‘evacuate Place’ movement. In the following cycle, these movements should occur in the reverse order, meaning Place should move above Scale (since the movement was triggered by Scale), after which Goal should be adjoined to Scale. However, this order, in which Place moves out of its constituent, is unlikely to be the right scenario. Therefore, I assume the opposite order, in which Goal move to Scale taking Place with it, after which Place moves above Scale.

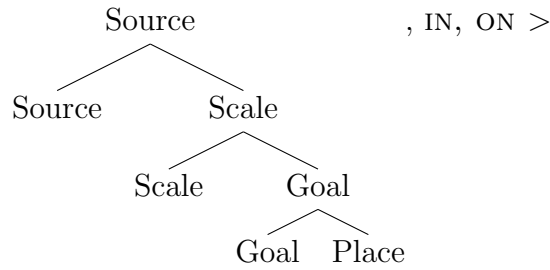
The ‘flicking’ Place feature of *spre* analysis is a possible way of addressing the interchangeability of *spre* and *inspre*. The order of *in* and *spre* emerges from the derivation, which is a welcomed result.

This analysis does not, however, provide a answer to the question of why the non-transitional Source Path expression is always built with *inspre*. Under the story presented above, this would imply that every time a Source Path is derived instead of a Goal Path, the Place bulb of *spre* is out and *in* surfaces.

(100)



It is unclear what would motivate the Place feature of *spre* to be off whenever a Source Path is constructed, but off or on when a Goal Path is. An alternative is to consider a ‘frozen’ lexical entry *dinspre* with the specification in (101).

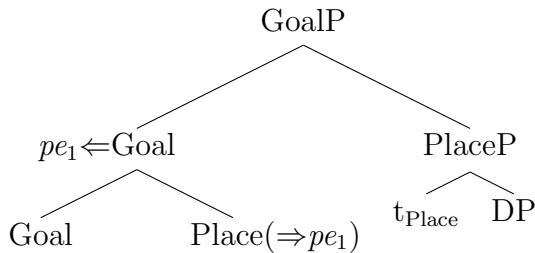
(101) *dinspre* ⇔ </dinspre/>,

While it may be on the right track, this solution seems quite inelegant within the present framework, since *dinspre* can clearly be decomposed into smaller units. For this reason, I hesitate to adopt it, and would argue for the need of an analysis that treats the obvious components of *dinspre* as separate lexical items.

## 5.2 *peste*

The specific transitional Route preposition *peste* of the ON series deserves some special attention. Let us start with a Transitional Goal Path, lexicalized by *pe*<sub>1</sub>, the counterpart of *la* of the AT series.

(102)

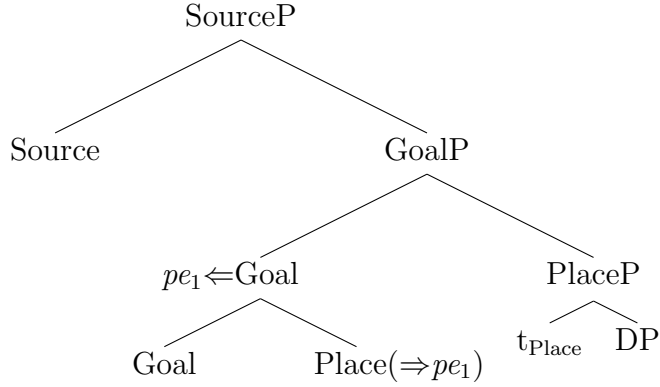


Granted, *peste* could lexicalize the Goal Path, but up to now it has not been chosen, due to the extra features it contains.

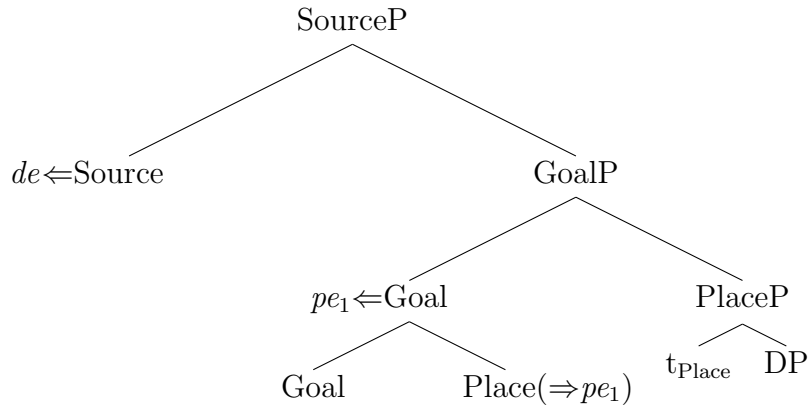
The following steps towards building a Route Path are illustrated below. Once we have the Goal Path, the next step is to add the Source feature.

(103) Transitional Route Path, the ON series

a. Step 1: merge Source

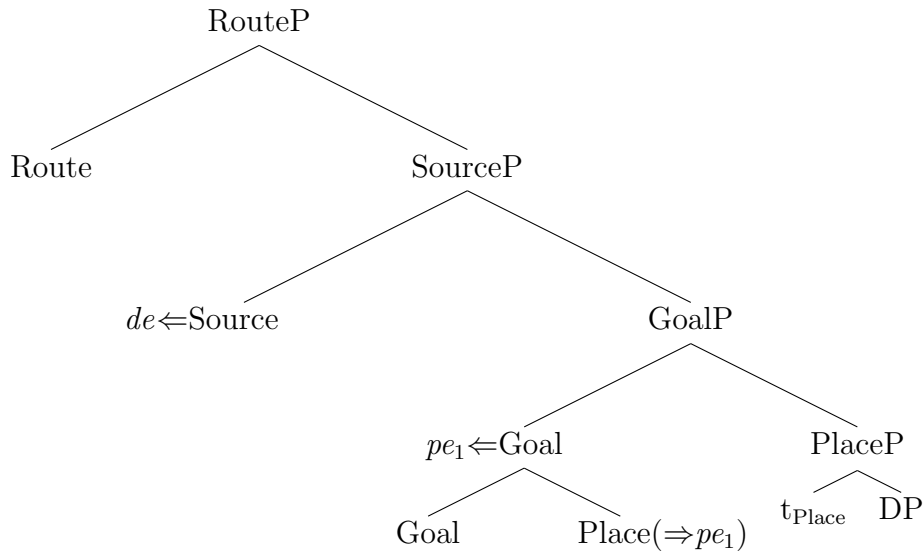


b. Step 2: inspect Source; insert *de*



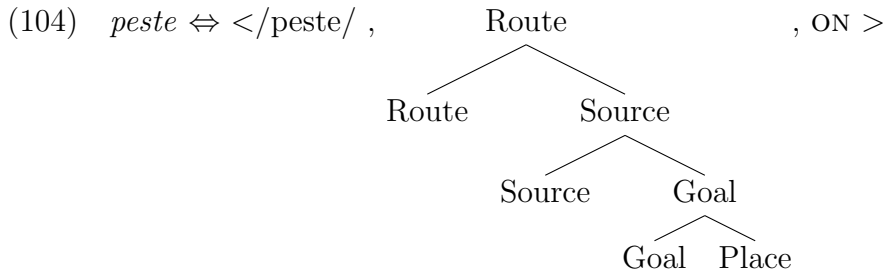
c. Step 3: inspect SourceP; no item is inserted; SourceP is lexicalized by inheritance

d. Step 4: merge Route



The next inspected node is Route. Two lexical items compete: *pe*<sub>2</sub> and *peste*. *pe*<sub>2</sub> contains the extra feature *Scale*, and *peste* does not, so the latter gets chosen.

We are now facing a problem. The data tell us that it is indeed *peste* which gets chosen, but in order to be able to insert this item, the syntactic configuration in its lexical specification must be created. Recall the entry for *peste*:



Since *peste* gets inserted at the Route node, it makes use of its entire lexical specification, for which the adjoining of Source, Route, and Goal is required. However, the structure we are left with after building the Source Path requires two moves in order for this configuration to be created. Up until now, the missing material needed to adjoin to a node already formed a single constituent at the time the movement was triggered, so the system only needed to move the topmost feature of the missing material, with the rest being pied-piped. In the case at hand, Source is alone, with Goal and Place forming a separate constituent. For a convergent derivation, we need all three in one constituent.

It may be reasonable to consider that, before inserting *peste* at Route, the system does not know or check whether the missing material is in one constituent. Once *peste* is inserted, the movements it triggers are Source to Route and Goal to Source. This could be done by going through the tree, top-to-bottom, and arranging for the adjoining of the needed features. At this point, the obvious question regards the relative ordering of the movements. Since the tree is searched top-to-bottom, the order in which the movements are triggered will be Source to Route first, and Goal and Place to Source second. As for the order in which movement operations are performed, there may be a reason to assume the same one as in the previous subsection. Suppose that *peste* needed only up to Goal. If Goal and Place form a constituent, Place will go along with Goal in the movement operation. Then, it would have to be evacuated. For the reason outlined in the previous subsection, evacuation should be triggered after addition of material, and should happen afterwards, after the constituent is formed. Thus, by generalizing this line of reasoning, in such cases, movement operations are performed in the order in which they are triggered.

## 6 Conclusion

This paper presented an analysis of Romanian directional expressions within a nanosyntactic framework, making use of the apparatus outlined in Pantcheva(2011). After a synopsis of said apparatus, data for four series were presented and, through systematization, a table of Romanian Path prepositions was obtained. The table was then used to determine what the span of each preposition is, after which the discussion proceeded to the shapes of the (syntactic subtrees of the) prepositions and the details of the derivation. Making use of the derivation process, it was argued that the prepositions do not have phrasal nodes, following Caha (2010). Finally, the steps for building each type of Path were illustrated, before discussing two problematic cases.

## REFERENCES

- CAHA, P. (2009). *The Nanosyntax of Case*. Ph.D. thesis, University of Tromsø.
- CAHA, P. (2010). *The Parameters of Case Marking and Spell-out driven movement*. In *Linguistic Variation Yearbook 2010*. John Benjamins.
- PANTCHEVA, M. (2011). *Decomposing Path. The Nanosyntax of Directional Expressions*. Ph.D. thesis, University of Tromsø.
- STARKE, M. (2009). *Nanosyntax: A short primer to a new approach to language*. In *Nordlyd 36.1: Special issue on Nanosyntax*, 1-6. Available at [www.ub.uit.no/munin/nordlyd](http://www.ub.uit.no/munin/nordlyd) .
- STARKE M. (2011). *Towards elegant parameters: Language variation reduces to the size of lexically stored trees*. Transcript from a talk at Barcelona Workshop on Linguistic Variation in the Minimalist Framework. Available at <http://ling.auf.net/lingBuzz/001183> .