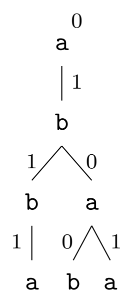
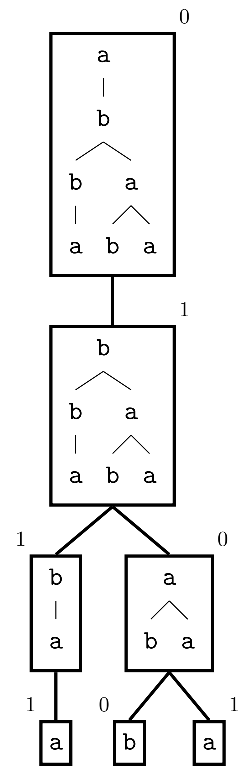
7. Minimalist Grammars (MGs)

1 Derivation trees

Thinking back to our "even number of as" FSTA, for example, we indicated the role of *states* as in (1). A more verbose way of expressing the same thing would be (2).

(1) 

(2) 

A representation like (2) is redundant here, because in the case of an FSTA we can always identify the **derivational antecedents** of a tree by looking at its **subparts**.

In MGs, an expression's derivational antecedents *do not* always coincide with its subparts, so representations like (2) are not redundant in this way.

2 MG basics

MGs were originally defined in Stabler 1997; a good overview is in Stabler 2011. Many of the examples here are from the introductory chapter of Graf 2013.

MGs are a completely lexicalized formalism: a grammar for a particular language just is a collection of lexical items. A small example grammar (based on Graf 2013, p.12) is given in (3).

(3)

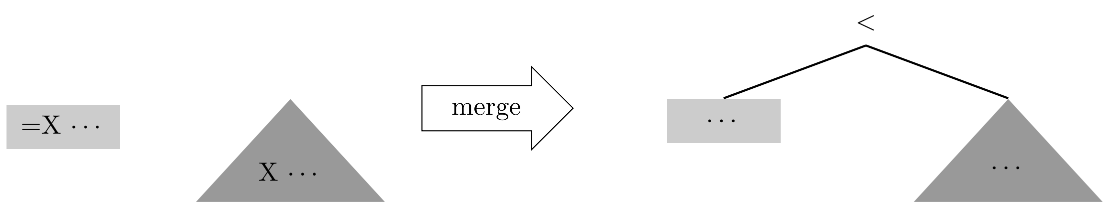
`the` :: =N D	`pigs` :: N	ε :: =V =D v
`the` :: =N D –nom	`sleep` :: V	ε :: =v +nom T
`which` :: =N D –wh	`kiss` :: =D V	`that` :: =T C
`which` :: =N D –nom –wh	`owe` :: =D =D V	ε :: =T C
`'s` :: =N =D D	`tell` :: =C =D V	ε :: =T +wh C

The pieces of information on each lexical item that specify what can combine with what are called **features**.

- Merge checks a *selector feature*, written =F, and a *category feature*, written F.
- Move checks a *licensor feature*, written +F, and a *licensee feature*, written –F.
- The head of each newly-formed constituent is the element that had a selector feature (in the case of merge) or licensor feature (in the case of move) checked.
- The features on a lexical item **must be checked in order** (starting with the leftmost). So bringing a lexical item into a derivation commits you to a certain ordered bundle of derivational operations: a merge step for each selector/category feature, and a move step for each licensor/licensee feature.

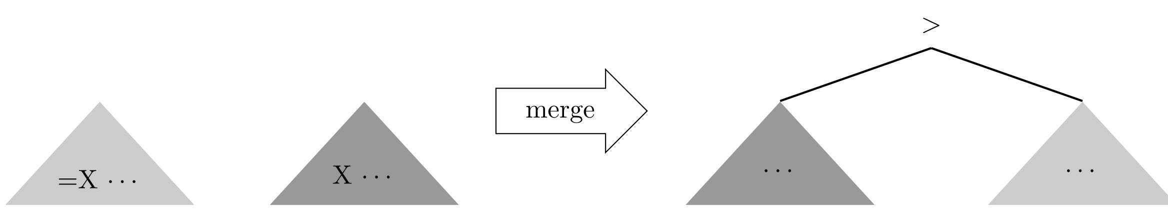
If the element having a selector (=X) feature checked by merge is a lexical item (i.e. a trivial, one-node tree), then it becomes the left daughter of the resulting tree.

(4)



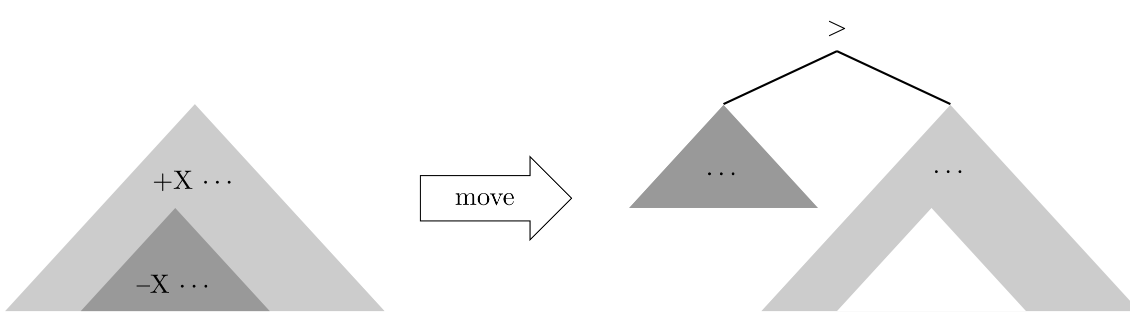
If the element having a selector (=X) feature checked by merge is a complex tree, then it becomes the right daughter of the resulting tree.

(5)



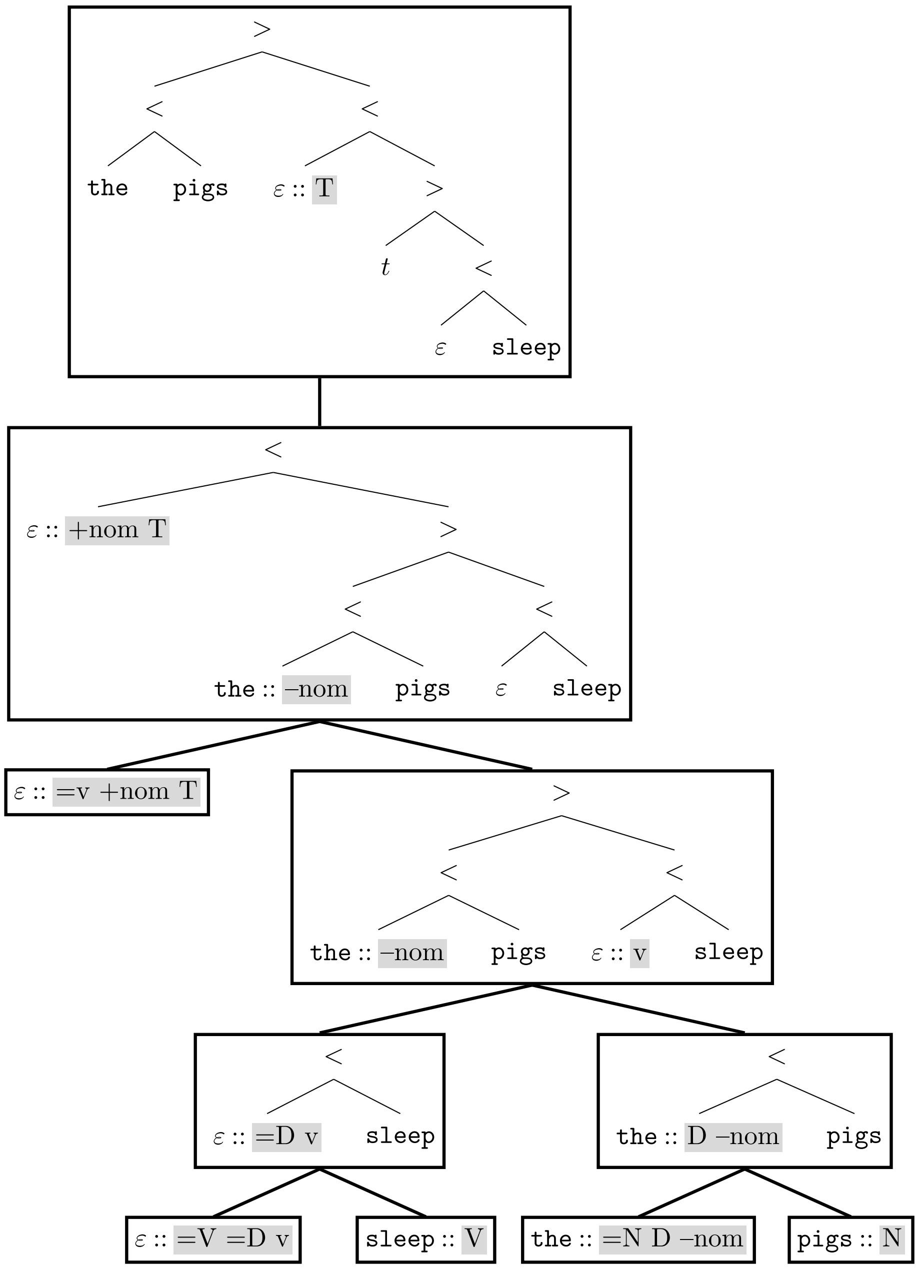
The element having a licensee (–X) feature checked by merge becomes the left daughter of the resulting tree.

(6)



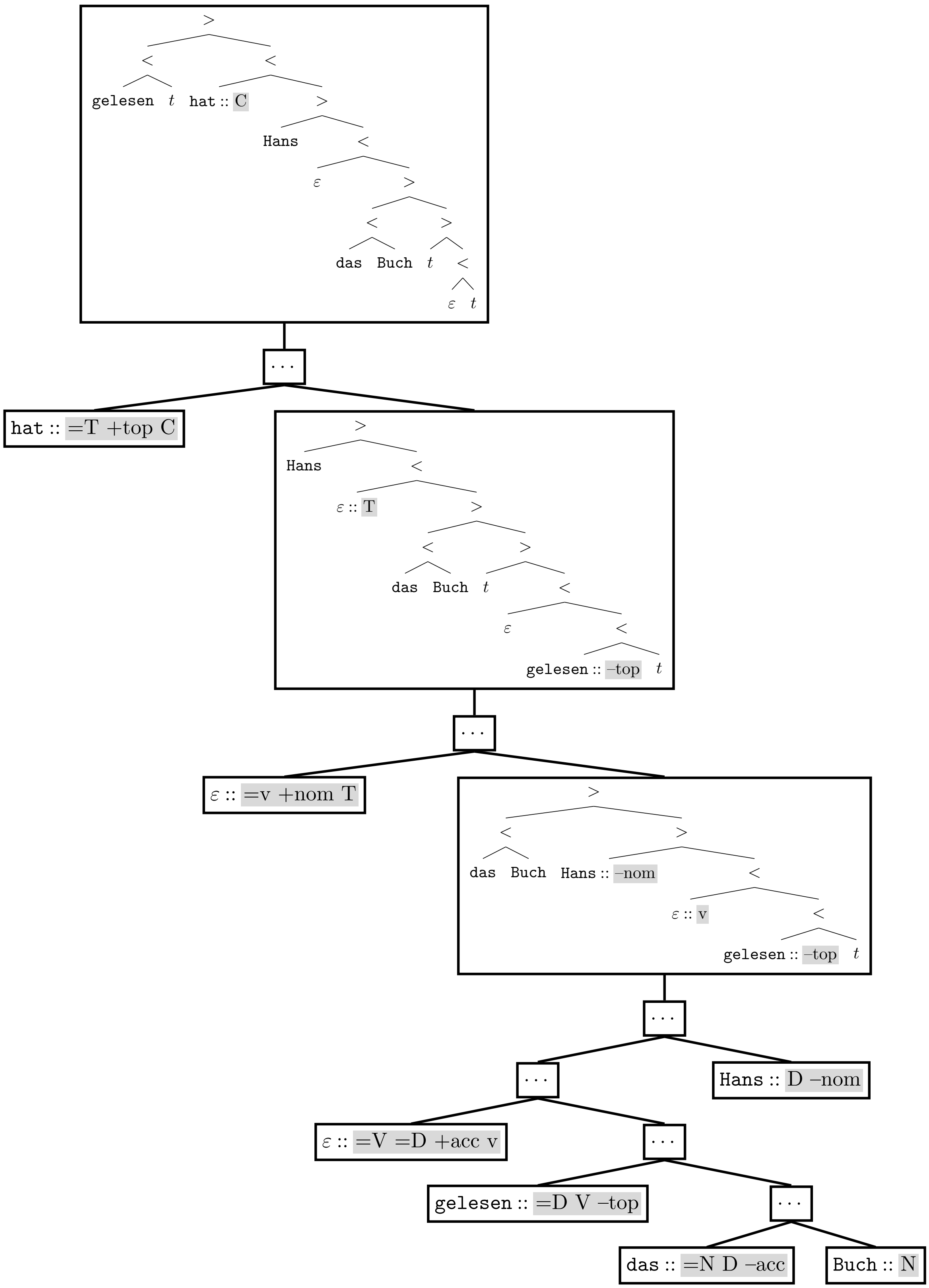
Here's one very simple derivation (from Graf 2013, pp.13–18).

(7)



Here's a more interesting example involving remnant movement (Graf, 2013, pp.19–20).

(8)



3 The Shortest Move Constraint (SMC)

- The SMC is often presented (e.g. Stabler, 1997, 2011) as a part of the definition of the move operation: move can apply to an expression iff
  - the first feature on the expression's head is a licensor feature, and
  - there is *exactly one* subconstituent whose first feature is a matching licensee feature.
- A consequence of this, however, is that if we ever find ourselves with two "competing" licensee features on subconstituents of a single expression, there will be no way to check either of them, and so such an expression is doomed. So the SMC can equivalently be stated as a ban on expressions where licensee features "compete" in this way (Graf, 2013, p.25).
- For example, if +/–nom and +/–acc in (8) were replaced by simply +/–case, we would end up with two competing –case licensee features after `Hans` is merged, and that derivation would be doomed.

4 Derivations are finite-state

The SMC guarantees that an MG's derivations are finite-state.

4.1 What information matters?

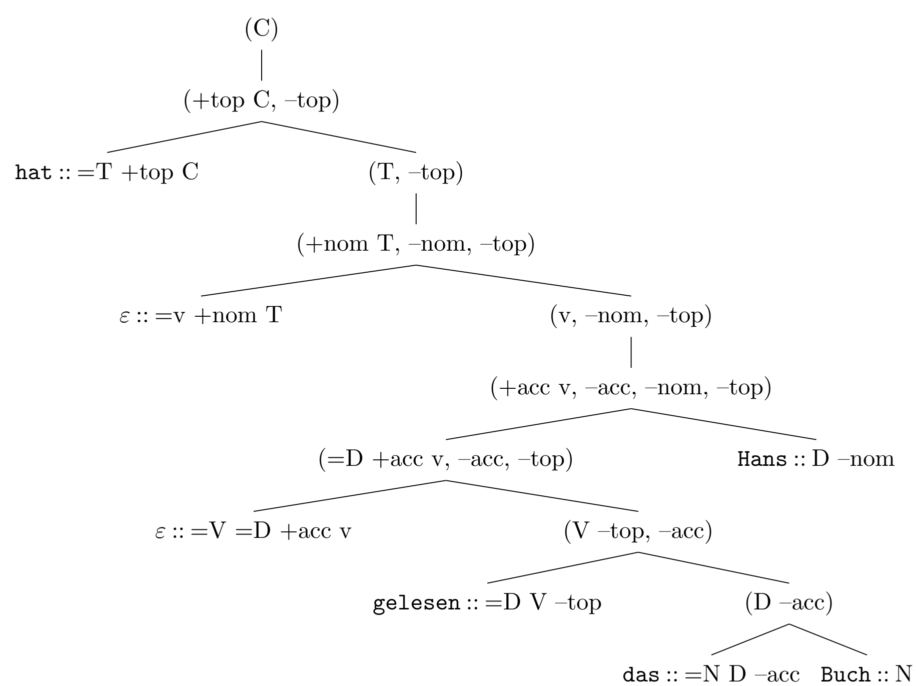
First, notice that it suffices to keep track of:

- the feature-sequence on the current head, and
- the feature-sequences on any other subconstituents.

The part that's slightly unintuitive, at first, is that it's not important to keep track of *where* the subconstituents with unchecked features are. The reason this is not important is that they *must* move out of their current positions.

Here's a tree which shows *just the states* that the derivation in (8) transitions through.

(9)



4.2 How many distinctions are there?

Then, the SMC ensures that only finitely many of these tuples-of-feature-sequences (i.e. states) are needed:

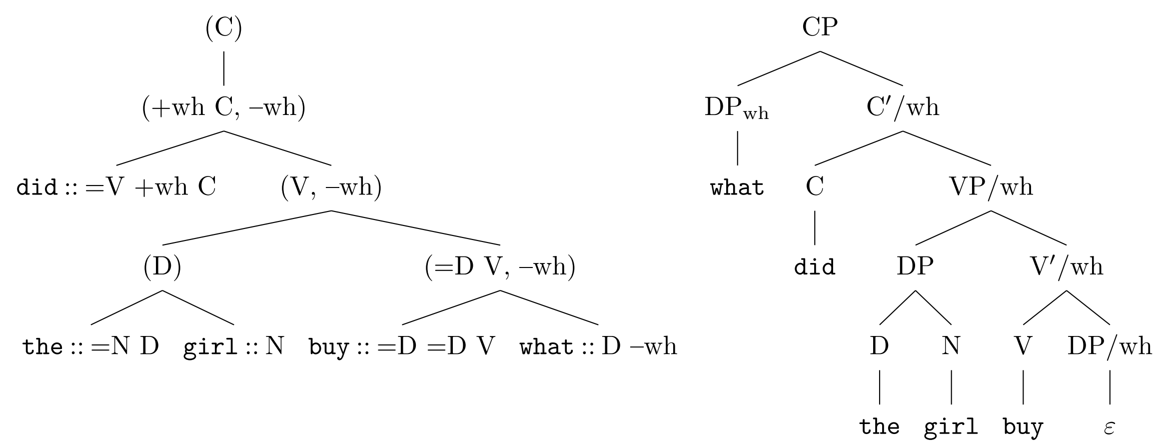
- Each component of the tuples is a suffix of a lexical item's feature-sequence, so there are only finitely-many possibilities in each slot.
- There is a finite bound on the number of slots — namely, the number of licensee feature types plus one.
  - If any component other than the initial component contains anything other than licensee features, the derivation is doomed.
  - If any two of these non-initial components both start with –F, for some particular feature type F, then the derivation is doomed.
  - So the biggest possible tuple contains an initial component, plus one additional component for each distinct type of licensee feature.

5 Where does the non-context-freeness come from, exactly?

Unsurprisingly, MGs without move generate only the context-free string languages: in such an MG, the finite-state derivational process doesn't "disrupt" existing tree structure at all, so it's essentially an FSTA.

More interestingly: MGs with only "one at a time" phrasal movement (e.g. if there is only one movement feature type) also only generate context-free string languages, because the effects of this limited kind of structural disruption can be mimicked by an FSTA/CFG.

(10)



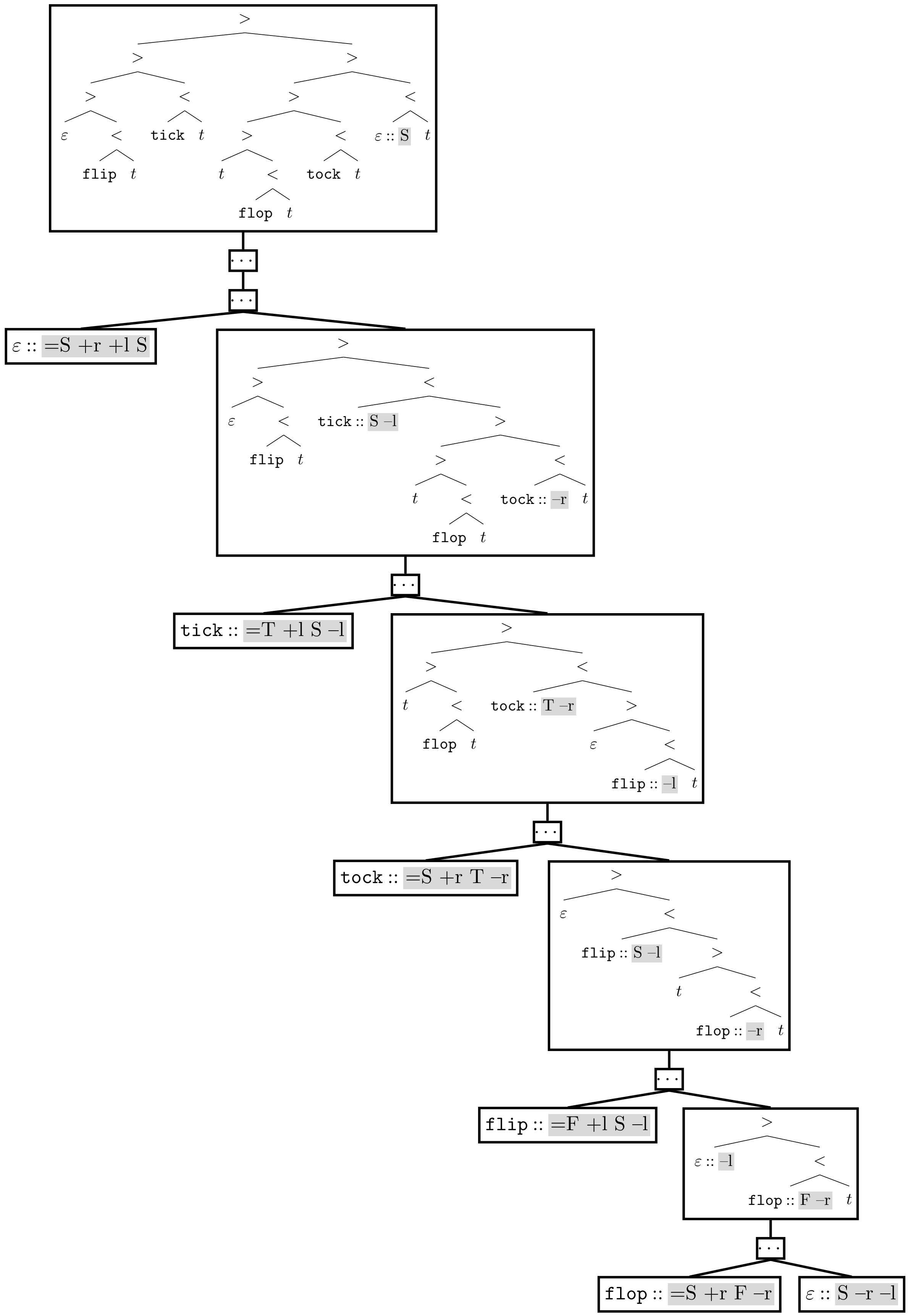
> For an MG to generate a non-context-free string language, it needs to allow the distortions created by movement to *accumulate*. Remnant movement or head movement can do this.

Here's a grammar that generates the `flip`/`flop`/`tick`/`tock` crossing-dependencies language, using remnant movement.

(11)

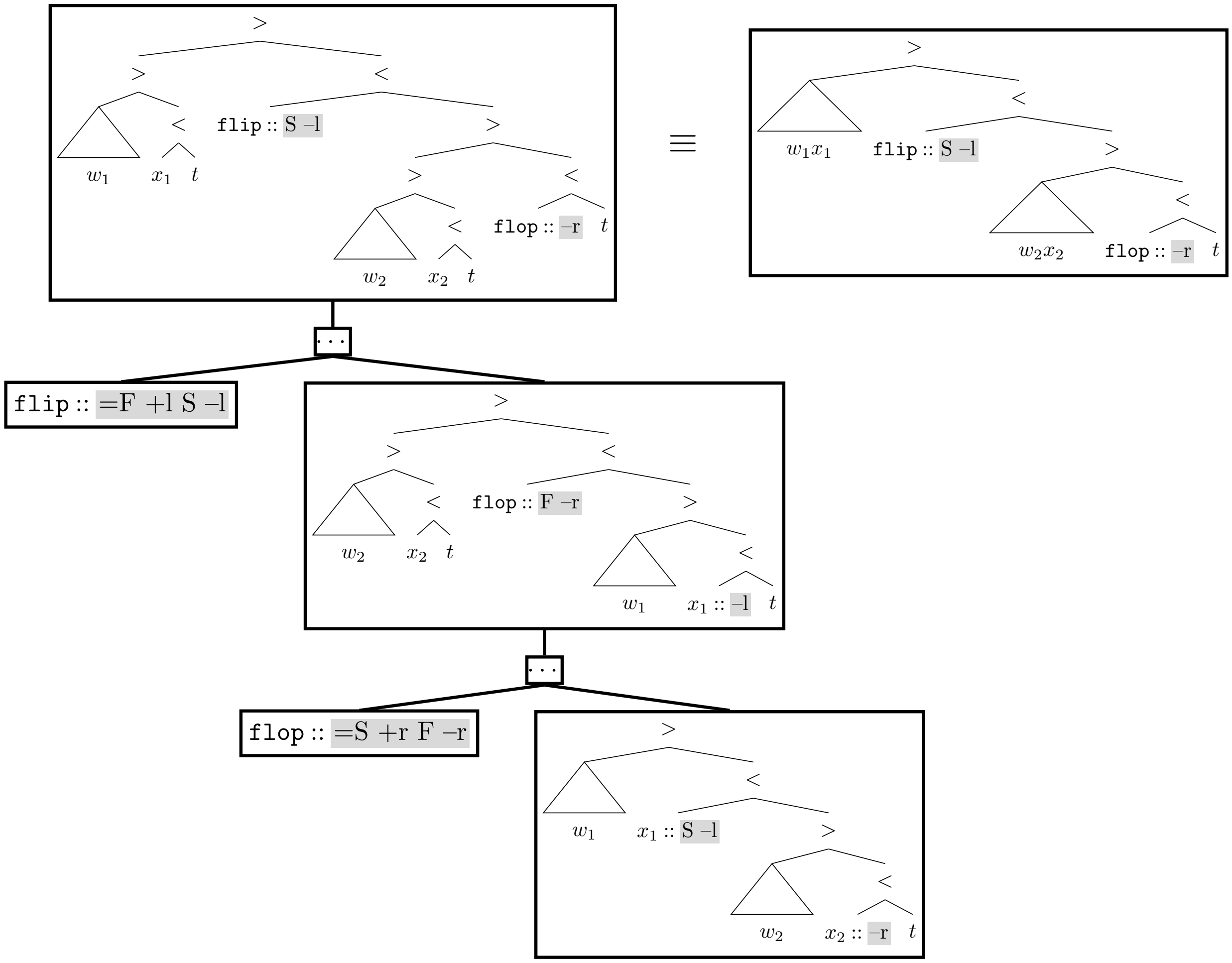
ε :: S –r –l	ε :: =S +r +l S
`flop` :: =S +r F –r	`tock` :: =S +r T –r
`flip` :: =F +l S –l	`tick` :: =T +l S –l

(12)

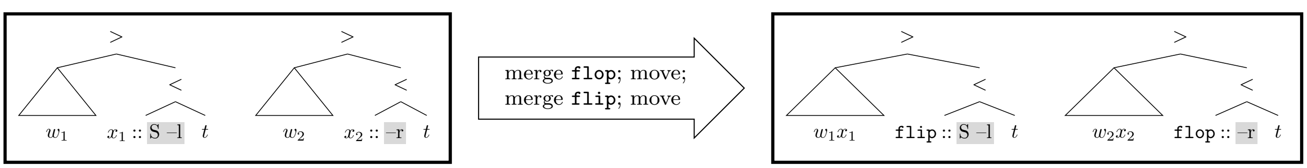


The key idea is that this grammar allows for "derivational loops" like this:

(13)



In effect, remnant movement allows us to work at the roots of two independent subtrees.



This is the fundamental difference that allows MGs to go beyond FSTAs: FSTAs categorize *individual trees* that grow bottom-up, whereas **MGs categorize *tuples of trees* that grow bottom-up**.

This also underlies the equivalence between MGs and *Multiple Context-Free Grammars* (MCFGs), which operate over *tuples of strings*; see e.g. Hunter and Dyer 2013, §2 for a brief explanation of this.

References

Graf, T. (2013). *Local and transderivational constraints in syntax and semantics*. PhD thesis, UCLA.

Hunter, T. and Dyer, C. (2013). Distributions on Minimalist Grammar derivations. In *Proceedings of the 13th Meeting on the Mathematics of Language*.

Stabler, E. P. (1997). Derivational minimalism. In Retoré, C., editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *LNCS*, pages 68–95, Berlin Heidelberg. Springer.

Stabler, E. P. (2011). Computational perspectives on minimalism. In Boeckx, C., editor, *The Oxford Handbook of Linguistic Minimalism*. Oxford University Press, Oxford.