Case Theory in Minimalist Grammars

Sabine Laszakovits

Department of Linguistics
University of Connecticut
sabine.laszakovits@uconn.edu

Abstract. This paper investigates the consequences of one-to-many licensing relationships for Minimalist Grammars (MGs; [30]) on the example of case. Dependent Case Theory [2, 23] has proposed that a single noun phrase can assign accusative case to arbitrarily many other noun phrases in particular structural configurations. Taking a licensing view rather than an assignment view on the distribution of case, this implies that accusative case can be licensed by a single licensor on arbitrarily many licensees. This paper argues that the distribution rules for case can be formalized as at most monadic second-order constraints, which are known to be translatable into an MG with refined Merge-features [16]. However, an implementation as Move-features is not feasible because such an MG would need to "count" and would thereby generate non-regular derivation tree languages. It is argued that this increase in complexity can be avoided by suspending the SMC for licensing relationships that involve neither displacement of phonological nor of semantic features.

Keywords: Dependent Case Theory · Minimalist Grammar · licensing · persistent features · SMC

1 Introduction

In natural language syntax, long-distance dependencies are sometimes formalized as covert movement. The licensee moves into the licensor's specifier position and establishes a local feature checking/valuation relationship without effects on the word order at PF, nor on the scope relations at LF. In some constructions, one licensor is involved in multiple dependencies of the same type, even in arbitrarily many. This paper explores one such construction in detail: accusative assignment in the framework of Dependent Case Theory (DCT). DCT argues that accusative is assigned to a noun phrase (NP) in the presence of another NP obeying certain structural configurations. Phrased differently, one NP can license accusative case on arbitrarily many NPs standing in these configurations. Other examples of one-to-many licensing relationships in natural language syntax include NPI-licensing, anaphor-binding, negative concord, agreement, parasitic morphology, and sequence of tense.

The questions that arise are the following:

1. What is the state of the derivation at the point of applying these movement operations?
   How does the derivation keep track of the arbitrary number of licensees?
2. What is the decision procedure to determine in what order the licensees move?

I will show that one-to-many licensing configurations cannot be computed within the power usually assumed for natural languages (regular tree languages; sec. 4). We do not have independent reason to justify the increase in power to context-free tree languages, which is necessary to capture these arbitrarily many movement steps. In fact, we can show that a formalization within regularity *is* possible as long as we do not resort to covert movement to formalize it, but employ MSO-definable constraints imposed by refined selection features [16] (sec. 3). This illustrates a fundamental asymmetry between selection and licensing: every constraint that is definable by MSO-logic can be expressed by selection, but not necessarily by licensing.

Minimalist Grammars [30] have formalized the restriction to regularity of their derivation trees into what is known as the Shortest Move Constraint (SMC) — a categoric constraint against arbitrarily many licensees simultaneously awaiting movement [29]. This paper exhibits a formal procedure to circumvent the SMC for particular feature checking relationships without increasing the necessary power of the formalism (sec. 5). Given that the licensing movement exhibited by DCT influences neither PF nor LF, we can avoid answering question 2. Rather, we can establish all required licensing relationships simultaneously, irrespective of their number. Thus the formalism does not need to remember the number of waiting licensees, answering question 1. At any given time, the only relevant information is whether or not *at least one* licensee is in need of licensing. Whenever a licensing relationship is established, *all* unchecked licensees will be checked.

This then establishes a way to formalize one-to-many licensing relationships with movement features in frameworks that hitherto were constrained by the SMC.

Predictions for wh-movement. Constructions with one-to-many licensing relationships requiring movement that influences PF have been discussed in the literature on the example of multiple wh-fronting [12, 13]. This phenomenon is outside the scope of this paper, but I do make a prediction about it: For constructions in which not all fronted wh-words stand in a c-command relation to another fronted wh-word (in their respective base positions), and not increasing the formalism's power, we predict that the order of moved wh-phrases is not determined by the derivation, i.e., it is either completely free (arbitrary) or completely fixed (determined by a deterministic post-syntactic mapping establishing linear order without reference to syntactic features). However, see [5] and references cited therein on the absence of wh-movement as well as wh-movement to lower positions.

1.1 Outline

This paper is structured as follows. In sec. 1.2, I introduce the basic idea behind Dependent Case Theory and illustrate the challenges raised by unbounded dependent-down case assignment. Section 2 defines Minimalist Grammars as regular tree grammars generating derivation tree languages [18, 30, 32]. The following three sections discuss possible implementations of MGs generating tree languages whose distribution of accusative case matches Dependent Case Theory. Section 3 proves that an implementation with a regular tree language is possible by giving an implementation using monadic second-order constraints. Section 4 shows that an implementation with long-distance licensing relationships is not possible without significantly increasing the complexity class of this tree language from a regular to a context-free tree language. Section 5 suggests an amendment: long-distance licensing relationships can be employed if we adapt the formalism so that dependent-case is only assigned to a single nominal, rather than to unboundedly many. Section 6 concludes.

1.2 Dependent Case

Dependent Case Theory (DCT; [23,34]) regulates how and when nominals receive morphological case-marking. At present, the rules that generate the distribution of case morphology in various works on DCT [2, 4, 21, 23, 27, 28] are given as high-level descriptions of the relevant configurations. For each case there are rules specifying the contexts in which it can be assigned. These rules fall into four categories, defining *inherent cases*, *dependent cases*, *unmarked cases*, and *default cases*, which in turn stand in a hierarchy (1) such that if a noun is eligible for more than one case, only the case in the highest-ranked (left-most) category will be assigned.

(1) inherent cases > dependent cases > unmarked cases > default cases [23]

Inherent cases (also: lexical, quirky, idiosyncratic cases) are assigned due to a fixed property of the element that introduces the nominal into the derivation. For example, in Icelandic the verb 'to help' always assigns dative to its helpee-argument, even under passivization, where non-inherently accusative marked arguments undergo case alternation to nominative. We use the term *structural cases* to refer to all non-inherent cases.

Dependent cases are structural cases that are assigned in configurations of two or more nominals in the same case assignment domain that stand in a c-command relationship. Dependent-down cases are assigned to the c-commanded nominal, and dependent-up cases to the c-commanding nominal. For example, accusative is a dependent-down case and applies in the CP domain. Dependent-up cases include ergative [3] in the CP domain, and dative in the VP domain [4].[1]

[1] Language differ with respect to restrictions on the licensing nominals, in particular with respect to their case marking. In many languages, nominals with inherent case cannot be licensors ("quirky subjects", e.g. in Icelandic [34], Diyari, Kannada [2]). If

Unmarked cases arise when dependent cases do not, i.e., in the absence of other elements in the structure, and they are sensitive to the case assignment domain. In most languages, nominative is the unmarked case in the CP domain. Genitive has been argued to be the unmarked case in the DP domain [2, 23].

Default cases arise outside of case assignment domains, such as in fragment answers or hanging topics. In many languages, default case is nominative, however, for English it has been argued that default case is accusative [23].

Algorithm An algorithm based on Marantz's [23] disjunctive case hierarchy, in more modern syntactic terminology of e.g. [2], is given in fig. 1. As soon as the

Precondition: α does not have case.
1. The lexical item that merges α can assign *inherent case* to α.
2. If α is still case-less, determine the case-assignment domain D that α is in.
   2.1. If there is another NP β in D that does not have inherent case, and β c-commands α, then assign to α the *dependent-down case* associated with D (if any).
   2.2. If α is still case-less and there is another NP γ in D that does not have inherent case, and α c-commands γ, then assign to α the *dependent-up case* associated with D (if any).
   2.3. If α is still case-less, assign it the *unmarked case* associated with D.
3. If α is still case-less (i.e., it is not in a case-assignment domain), assign it *default case*.

Postcondition: α has case.

Fig. 1. Algorithm for Case Assignment in DCT

noun phrase (NP) α receives case, the algorithm is exited, leading to at most one case per NP per case-assignment domain. Together with the algorithm's assertion that every NP will receive case[2], this leads to an assignment of *exactly* one case per NP per case-assignment domain. The rule ordering of dependent-down case before dependent-up case captures the fact that in a sequence of three NPs α, β, γ such that α c-commands β, and β c-commands γ, all in the same domain D associated with both a dependent-up case (e.g., ergative) and a dependent-down case (e.g., accusative), NP β receives accusative rather than ergative; [2, 232] on Diyari.

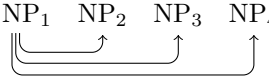
---

the subject carries dative, the object will carry nominative, not accusative. However, Tamil as well as some dialect of Faroese exhibit DAT–ACC patterns [2, 187–194], and some dialects of Kurdish allow ERG-marking on the subject if the object carries inherent DAT [1], as is also found in Warlpiri, Burushaski, and Ingush [2, 187–194]. In this paper, I will model the Icelandic patterns, but everything I say extends straight-forwardly to the Faroese pattern.

[2] It has been argued [20, 21] that unmarked cases are the morphological marking that arises in the absence of case. In this paper, I take unmarked cases to be assigned and licensed like other cases.

Dependent-down case licensing is unbounded. Dependent-down case licensing has a property that the other categories of case do not share. A single NP can assign dependent-down case to unboundedly many nominals in its c-command domain. In (2), arrows indicate assignment of accusative case.

(2) $\mathrm{NP}_1 \quad \mathrm{NP}_2 \quad \mathrm{NP}_3 \quad \mathrm{NP}_4 \quad \cdots$



In principle, there are two kinds of configurations for such a series of NPs to stand in. The first is pairwise c-command relations: NP_1 c-commands NP_2, NP_2 c-commands NP_3, NP_3 c-commands NP_4 etc. Asymmetrical c-command is transitive: it follows that NP_1 also c-commands NP_3 and NP_4, that NP_2 also c-commands NP_4 etc. We thus have multiple potential case assigners for NP_i, $i > 2$: $\mathrm{NP}_1, \ldots, \mathrm{NP}_{i-1}$. We will refer to this configuration as *daisy-chain* configuration.

The second configuration contains NP_2 to NP_n in such positions that for all i: NP_i does not c-command any NP in the above sequence. Thus all dependent-down case-marked NPs in this sequence only have one potential case assigner: NP_1. We will refer to this construction as *1:n*-configuration. Clearly, combinations of these two configurations are also possible.

The daisy-chain configuration is well-attested, for example in Finnish [22,26]. In a sequence like (2), we can observe that removal of NP_1 results in NP_2 changing its case from accusative to nominative, but all following NPs retain accusative case. Similarly, additional removal of NP_2 changes the case on NP_3 from accusative to nominative, but the case on NP_4 etc. remains unchanged. Under the assumption that the structures with NP_1 and without NP_1 are identical in all relevant aspects except for the presence or absence of NP_1, this is evidence for NP_{i+1} receiving its accusative case from NP_i, $i > 1$, and not from NP_1.[3] The daisy-chain configuration is directly expressable by regular tree grammars using licensor features for case and thereby does not pose a problem for the formalism.

The 1:*n*-configuration is attested in Sakha [4,33]. In "raising to object" constructions, the subject of an embedded clause (finite or nominalized) can receive accusative marking.

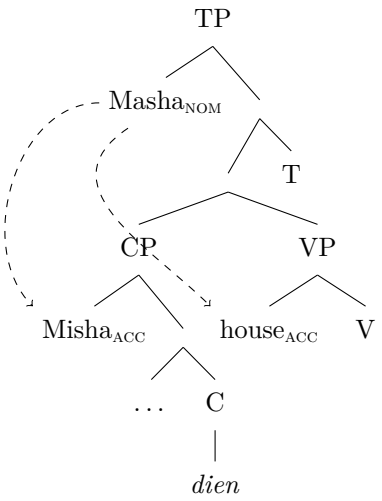
(3) *Masha* [*Misha-ny* *kel-ie* *dien*] *djie-ni* *xomuj-da.*
Masha.NOM Misha-ACC come-FUT.3SG that house-ACC tidy-PST.3SG
'Masha tidied up the house (thinking) that Misha would come.' [33, 368]

Baker and Vinokurova [4, §3.5] show that the availability of this accusative marking is dependent on the presence of a matrix subject, and not on other sources of accusative marking (such as functional heads) in either the embedded clause or the matrix clause. When changing the matrix predicate to subject-less

[3] [20,21] argue that only case-less NPs (carrying unmarked case) can license dependent case. If this is correct, the daisy-chain configuration for case assignment does not exist, and these structures employ the 1:*n*-configuration instead where all assignment is performed by NP_1.

predicates like 'it became certain' or 'it became necessary', the embedded subject cannot carry accusative, and the matrix object cannot carry accusative.[4] This then supports an analysis like (4), where the source of both accusatives is the matrix subject.

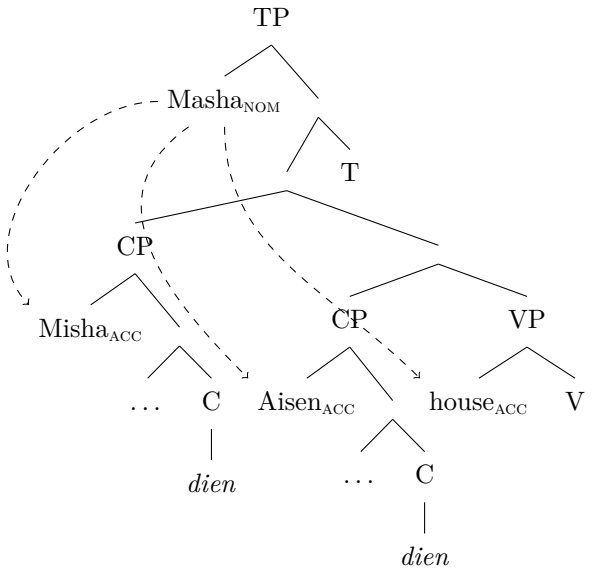
(4)



There remains the empirical question whether Sakha allows arbitrarily many instances of raising to object in one sentence. Given that *dien*-clauses are probably adjuncts, it is not unreasonable to think that there could be more than one in a single clause. Such a configuration is illustrated in (5).

[4] They do not, however, provide an example showing that both NPs lose their accusative-marking together.

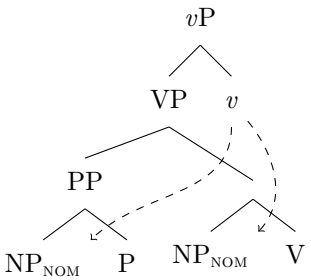
(5)



This paper does not aim to contribute to this empirical question and will instead assume DCT as formalized in the literature: (4) extends to configurations with unboundedly many adjunct-clauses, whose subject can receive accusative case that is licensed by the matrix subject.

A similar issue with unmarked case. Similar to dependent-down cases, a single clause can contain unboundedly many NPs carrying unmarked case. Unlike dependent-down cases, unmarked cases do not have an obvious licensor, however they are dependent on the case-assignment domain. Formally, we can interpret this as the head of this domain (for [2]: the phase head) licensing unmarked case. Then the treatment of unmarked case becomes parallel to that of dependent-down case. This is illustrated in (6), after a construction discussed in [2, 86]: Amharic dyadic unaccusatives (possessor or experiencer constructions) as well as passives of triadic verbs with a goal-argument. Since the two NPs in (6) do not stand in a c-command relationship, both are realized with unmarked case.

(6)



Case as Agree. An alternative option arises from feature valuation and Late Insertion: unmarked (and also default) cases are the elsewhere-forms of nominals that arise when lexical and dependent cases are not available. Instead of having nominals with unmarked case enter a licensing relationship with the head of the case assignment domain, we can make use of this relationship indirectly. Ermolaeva [11] has established a formal framework for Agree operations. Feature values can be transmitted along branches in a tree, 'riding' on top of merge- and move-operations. Nominals will carry a feature for the unmarked case in their case assignment domain, to be valued indirectly by the domain head. Following [2], i.a., I take case assignment domains to correspond to the complement of phase heads. A C-head will value the features on its complement as nominative (7a), and a D-head as genitive (7b). The outgoing information is specified as a superscript on `=T` resp. `=N` together with a right arrow.

(7) a. $[\ \varepsilon :: \texttt{=T}^{[U:\textsc{nom}]\rightarrow}\ \texttt{C}\ ]$ b. $[\ \varepsilon :: \texttt{=N}^{[U:\textsc{gen}]\rightarrow}\ \texttt{D}\ ]$

We specify all lexical items to percolate this morphological information down the tree, as illustrated in (8). Every lexical item will receive the value for unmarked case from its selector `Z` with a left arrow and pass it on to all LIs it selects: `=X` and `=Y` in (8). Simplifying [11], (8) uses the variable α to indicate that the value received via `Z` is the same as the value outgoing via `X` and `Y`.

(8) $[\ \begin{bmatrix}\phi\\ U:\alpha\end{bmatrix}\ ::\ \texttt{=X}^{[U:\alpha]\rightarrow}\ \texttt{=Y}^{[U:\alpha]\rightarrow}\ \texttt{Z}_{\leftarrow}\ ]$

NPs will receive the value for U from their selector as well. They are the only LIs whose phonological realization is sensitive to the value of U. If they have received NOM as value of U, they are pronounced as such, if they have received GEN as genitive, and if U has not been valued, as default case. Since values are transmitted on top of Merge and Move operations, we can straightforwardly use Agree to assign a value for inherent cases, which are selected for directly. However, dependent cases are transmitted at a distance, so they require discussion.[5] The valuation for dependent cases either has to proceed via Move-features, which, as we will see, constitutes a problem, or via refined Merge-features that are sensitive to the presence of the licensor. We will see in sec. 3 details about such a refinement for a licensing account.

An alternative might be to transmit the hypothetical value D for dependent case from the phase head across the entire case assignment domain, like unmarked case above, and then to introduce a spell-out rule that states that if an NP α has an admissible c-commander, it will spell out its hypothetical dependent case D, and if it doesn't, its hypothetical unmarked case U. I leave an investigation into the possible advantages of either approach for future research. Either way, if unmarked cases are a 1:n-challenge in the same way as dependent-down cases, the present treatment of dependent-down cases will extend to them. On

[5] [28, 205] brings up the idea that dependent cases are feature values that are assigned in the presence of the licensing NP, but does not spell out how this might proceed.

the other hand, if unmarked cases can be solved by different means, the present treatment of dependent-down cases is not affected.

2 Preliminaries

Our data structure are labeled trees $t = \langle V, \trianglelefteq, \mathit{label}\rangle$ over a ranked alphabet $\langle \Sigma, \mathit{Arity}\rangle$ such that V is a set of nodes (vertices), $\trianglelefteq$ (*dominance*) is a partial order on V, from which derive the predicates *strict dominance* $\triangleleft$ (for all $u, v \in V : u \triangleleft v$ iff $u \trianglelefteq v$ and $\exists w \in V : u \trianglelefteq w$ but not $v \trianglelefteq w$) and *immediate dominance* $\blacktriangleleft$ (for all $u, v \in V : u \blacktriangleleft v$ iff $u \triangleleft v$ and $\nexists w \in V : u \triangleleft w$ and $w \triangleleft v$), and *label* is a function $V \to \Sigma$ such that for all $v \in V, \sigma \in \Sigma$: $\mathit{label}(v) = \sigma$ only if the number of nodes that v immediately dominates $|\{v' \in V : v \blacktriangleleft v'\}|$ is equal to $\mathit{Arity}(\sigma)$. Two additional conditions must hold for t: the root condition: $\exists r \in V : \forall v \in V : r \trianglelefteq v$, and the chain condition: $\forall u, v, w \in V : u \trianglelefteq w$ and $v \trianglelefteq w$ implies that either $u \trianglelefteq v$ or $v \trianglelefteq u$ [17].

Following [18], we define Minimalist Grammars (MGs) as regular tree grammars G that generate derivation tree languages MDTL(G), which are subsequently mapped to various output structures by using monadic second-order transductions to specify final landing sites of moved elements, ordering of siblings, and headedness [16, §1.2.3]. For instance, they can be mapped to phrase structure trees, multi-dominance trees, strings, and many other representations.

We describe MGs by specifying their non-branching terminal nodes (their *lexicon* *Lex*) and the distribution of their branching terminal nodes with respect to *Lex*. A lexicon *Lex* is a finite set of triples $l = \langle p, s, f_1 \cdots f_n\rangle$ (*lexical items*, LIs) containing phonological content p (not to be specified further), semantic content s (not to be specified further), and a finite sequence $f_1 \cdots f_n$ of syntactic features, $n \geq 1$. We write lexical items as $[\ p :: f_1 \cdots f_n\ ]$, omitting the semantic content, with $\mathit{exponent}(l) = p$ and $\mathit{features}(l) = f_1 \cdots f_n$. A syntactic feature f has a type τ and a polarity π. Its type τ classifies f as either selection feature $\tau \in \mathbf{sel}$ or licensing feature $\tau \in \mathbf{lic}$, thus **sel** and **lic** are disjoint. Its polarity π is either positive or negative. The set *Feat* of syntactic features of a lexicon *Lex* is $\mathit{Feat} = \{\langle \tau, \pi\rangle : \langle \tau, \pi\rangle \in \mathit{features}(l), l \in \mathit{Lex}, \pi \in \{+, -\}\}$. For a syntactic feature $f = \langle \tau, \pi\rangle$, we use the following notations and designations:

	$\pi = +$	$\pi = -$
$\tau \in \mathbf{sel}$	=τ, "selector"	τ, "category"
$\tau \in \mathbf{lic}$	+τ, "licensor"	-τ, "licensee"

The sequence of syntactic features on a lexical item is always such that (i) all positive features precede all negative features, (ii) there is exactly one negative selection feature, (iii) the negative selection feature precedes all other negative features [14, 19].

A derivation tree $t = \langle V, \trianglelefteq, \mathit{label}\rangle$ will be in MDTL(G) depending on the syntactic features on its lexical items (its leaves), the distribution and labels of interior nodes, and well-formedness condition on its root. Nodes labeled Merge and Move introduce associations between two syntactic features of the same type

but opposite polarity on two different lexical items. A Merge node m is *projected* by a feature `=f` on a lexical item l_1 dominated by one of its two children and is an *occurrence* of a feature `f` of the same type but opposite polarity on a lexical item l_2 dominated by its other child. A Move node m is projected by a feature `+g` on a lexical item l_1 dominated by its single child c and is an occurrence of a feature `-g` of the same type but opposite polarity on a lexical item l_2 also dominated by c. There are locality restrictions. Informally, the positive features on an LI must project in order (i.e., for m_i the interior node projected by f_i, $m_{i+1} \triangleleft m_i$) and negative features must find the closest matching occurrence.

The well-formedness conditions on the root of t state that all features on all lexical items in t must have been associated with an interior node, except for one category feature `z` which must be in a given set F of allowed non-associated features.

Constraints on the distribution of terminal nodes follow from the feature calculus on LIs developed by [18, 30, 32]. A function h maps each node $v \in V$ in t to a tuple of feature sequences representing all the non-associated features on nodes dominated by v. If v is a lexical item, it does not dominate any nodes other than itself, and $h(v)$ only contains v's own feature sequence. If v is an interior node, h manipulates the feature sequences of v's children depending on v's label in the following way:

Definition 1 (Merge). *Given a node $v \in V$ with label$(v) =$ Merge, and two nodes $u, w \in V$ such that $v \blacktriangleleft u$ and $v \blacktriangleleft w$ and $h(u) = \langle \texttt{=x}f_2 \cdots f_k, \phi_1, \ldots, \phi_l \rangle$ ($\texttt{x} \in$ **sel**, $f_i \in$ Feat for $1 \leq i \leq k$ (i.e., the sequence $f_2 \cdots f_k$ is potentially empty), $\phi_i \in$ Feat* for $0 \leq i \leq l$) and $h(w) = \langle \texttt{x}g_2 \cdots g_m, \psi_1, \ldots, \psi_n \rangle$ ($g_i \in$ Feat for $1 \leq i \leq m$, $\psi_i \in$ Feat* for $0 \leq i \leq n$), then $h(v) = merge(h(u), h(w))$ combines $h(u)$ and $h(w)$ into a single sequence of sequences of features as defined below:*

$$\frac{\langle \texttt{=x}f_2 \cdots f_k, \phi_1, \ldots, \phi_l \rangle \qquad \langle \texttt{x}g_2 \cdots g_m, \psi_1, \ldots, \psi_n \rangle}{\langle f_2 \cdots f_k, \phi_1, \ldots, \phi_l, g_2 \cdots g_m, \psi_1, \ldots, \psi_n \rangle}\ merge$$

*If the initial features of the initial sequences of both $h(u)$ and $h(w)$ don't match such that they are of the same type $\texttt{x} \in$ **sel** and differ in their polarity, merge$(h(u), h(w))$ is undefined.*

Definition 2 (Move). *Given a node $v \in V$ with label$(v) =$ Move, and a node $u \in V$ such that $v \blacktriangleleft u$ and $h(u) = \langle \texttt{+x}f_2 \cdots f_k, \phi_1, \ldots, \phi_{i-1}, \texttt{-x}g_2 \cdots g_l, \phi_{i+1}, \ldots, \phi_n \rangle$ ($\texttt{x} \in$ **lic**; $f_j, k, g_j, l, \phi_j, n$ as above) such for all $j : 1 \leq j \leq n, j \neq i$: ϕ_j's initial feature is not `-x`, then $h(v) = move(h(u))$ is defined as given below:*

$$\frac{\langle \texttt{+x}f_2 \cdots f_k, \phi_1, \ldots, \phi_{i-1}, \texttt{-x}g_2 \cdots g_l, \phi_{i+1}, \ldots, \phi_n \rangle}{\langle f_2 \cdots f_k, \phi_1, \ldots, \phi_{i-1}, g_2 \cdots g_l, \phi_{i+1}, \ldots \phi_n \rangle}\ move$$

If the initial feature of $h(u)$'s initial sequence is not a licensor feature `+x`, and if there is not exactly one sequence of features in $h(u)$ whose initial feature is a licensee feature `-x` of the same type, move$(h(u))$ is not defined.

These definitions do not make reference to the ordering of phonological features on LIs. I assume that all linear order is established by post-syntactic mapping to derived structures.

The root condition can then be stated as: for r the root node, $h(r) = \langle \mathtt{z}, \varepsilon, \ldots, \varepsilon \rangle$ such that $\mathtt{z} \in F$.

Definition 3 (MG). *An MG $G = \langle N, T, S, P \rangle$ is a regular tree grammar with $T = Lex \cup \{\text{Merge, Move}\}$. The non-terminals N are tuples of sequences of features occuring in Lex, $N \subset (Feat^+)^+$, namely $N = \{\langle s_0, \ldots, s_{|\mathbf{lic}|} \rangle : s_i \in \{\beta : \alpha\beta = features(l), l \in Lex\}, 0 \leq i \leq |\mathbf{lic}|\}$, i.e., all $(|\mathbf{lic}| + 1)$-tuples made up of feature sequences that occur as suffix of the feature sequence of some lexical item. The start symbol $S = \langle \mathtt{C}, \varepsilon, \ldots, \varepsilon \rangle \in N$, $\mathtt{C} \in F$. The production rules P are defined based on the feature calculus operations merge and move; for $A_i, A_j, A_k \in N$:*

1. $A_i \to \text{Merge}(A_j, A_k)$ *if* $A_i = merge(A_j, A_k)$
2. $A_i \to \text{Move}(A_j)$ *if* $A_i = move(A_j)$
3. $A_i \to l$ *if* $A_i = features(l)$, $l \in Lex$,

where $X(Y, Z)$ stands for $X \blacktriangleleft Y$ and $X \blacktriangleleft Z$.

We now turn to implementing DCT as further constraints on MGs, which as [16] has shown can be implemented as refined **sel** features iff the constraint is MSO-definable.

3 Dependent-down case is regular

This section shows that dependent-down case can be formalized as MSO-constraint, thereby proving that an MG with Dependent Case Theory falls into the same complexity class as a standard MG: the class of regular tree grammars.

3.1 Defining the MSO-logic

We define an MSO-logic for derivation trees $t = \langle V, \trianglelefteq, label \rangle$ as generated by an MG G. To do so, we define a model structure $\mathcal{D} = \langle D, F, P^1, P^2 \rangle$ containing a domain D, a set F of functions on D, a set P^1 of first-order predicates, and a set P^2 of second-order predicates whose domain are tuples of unary first-order predicates; a signature $\Sigma = \langle FS, PS^1, PS^2, VS \rangle$ of function symbols FS, first-order predicate symbols PS^1, second-order predicate symbols PS^2, and variable symbols VS; and a signature interpretation Φ, which maps $FS \to F$, $PS^1 \to P^1$, and $PS^2 \to P^2$. Our domain D is the set of nodes V in t. P^1 contains the characteristic functions of $\trianglelefteq$ and $\blacktriangleleft$ such that $\trianglelefteq^{\uparrow} : (x, y) \mapsto \mathbf{t}$ if $x \trianglelefteq y$, $\mathbf{f}$ else; and $\blacktriangleleft^{\uparrow}$: $(x, y) \mapsto \mathbf{t}$ if $x \blacktriangleleft y$, $\mathbf{f}$ else; whose symbols are $\trianglelefteq$ and $\blacktriangleleft$, respectively. Furthermore, for every label σ in the range of $label$, P^1 contains a predicate labeled $Label_\sigma$ which identifies the nodes that have this label:

(9) $\Phi(Label_\sigma) : n \mapsto \mathbf{t}$ if $label(n) = \sigma$, $\mathbf{f}$ else.

Similarly, P^1 contains for each feature f a predicate labeled $Feat_f$ identifying the nodes labeled with lexical items that contain this feature.

(10) $\Phi(Feat_f) : n \mapsto$ **t** if $label(n) = l$ for some $l \in Lex$, and $f \in features(l)$; **f** else.

The predicate labeled *Occ* holds of a branching node m and a non-branching node n iff m is an occurrence of some feature on the label of n.

(11) $\Phi(Occ) : (m, n) \mapsto$ **t** if $label(n) = l$ for some $l \in Lex$, and m is an occurrence of some feature f in l; **f** else.

Furthermore, P^1 contains predicates labeled $assigns_{c,i}$ for each inherent case c in the natural language under discussion and for each integer i between 1 and the maximal number of positive features on LIs in the range of *label*.

(12) $\Phi(assigns_{c,i}) : n \mapsto$ **t** iff n is a non-branching terminal node labeled with some $l \in Lex$, and l has at least i positive features, and l's i-th positive feature f_i is $\in$ **sel**, and l assigns inherent case c via f_i; **f** else.

P^1 and P^2 contain predicates for all possible extensions. For all subsets $A \subseteq D$, P^1 contains a predicate $A^\uparrow$ identifying the nodes in A, and for all subsets $B \subseteq P^1$ of unary first-order predicates, P^2 contains a predicate $B^\uparrow$ identifying the unary first-order predicates in B.

The function labeled *Sliceroot* takes a node n that is either labeled with a lexical item l or projected by a lexical item l, and returns the highest node that l projects. For `c` l's unique category feature:

(13) $\Phi(Sliceroot) : n \mapsto$ the Merge-node projected by $\texttt{=f}_k$ if $label(n) = [\ p :: \ldots\ \texttt{=f}_k\ \texttt{c}\ \ldots\ ]$;
the Move-node projected by $\texttt{+f}_k$ if $label(n) = [\ p :: \ldots\ \texttt{+f}_k\ \texttt{c}\ \ldots\ ]$;
n if $label(n) = [\ p :: \texttt{c}\ \ldots\ ]$;
undefined else.

For a full definition see [15, 61].

Furthermore, for each integer i between 1 and the maximal number of positive features on LIs in *Lex*, F contains a function labeled $Proj_i$ that takes a non-branching node n and returns the branching node m projected by n's i-th positive feature:

(14) $\Phi(Proj_i) : n \mapsto$ the Merge/Move-node projected by n's i-th positive feature if n is labeled with an LI that has at least i positive features, **undefined** else.

It follows that for all nodes n, m, $Proj_0(n) = n$, and $Proj_{i+1}(n) = m$ iff $m \blacktriangleleft Proj_i(n)$ and n's label has at least $i + 1$ positive features.

3.2 Defining shorthand predicates

Before giving the full MSO-constraint capturing the rules of Dependent Case Theory, we will introduce some shorthand predicates (inspired by "pseudo-features" in [14, 16]). These are not predicates in the model structure $\mathcal{D}$, but abbreviations of MSO-formulas that serve no other purpose other than to improve readability.

First, we define three index sets stating the different cases for the natural language that we model. For example, in Turkish,

(15) a. $\textsc{Inh} = \{\textsc{abl}, \textsc{dat}, \textsc{loc}\}$
b. $\textsc{Dep} = \{\textsc{acc}, \textsc{dat}\}$
c. $\textsc{Unm} = \{\textsc{nom}, \textsc{gen}\}$

Note that dative (DAT) is both an inherent and a dependent case. This is to demonstrate that this formalism can deal with such ambiguities and with systematic syncretism in general.

We also specify which case is the default case in this language. For Turkish, $\textsc{def} = \textsc{nom}$.

Second, for each case c we specify a set $compat_c$ of LIs whose exponent is compatible with this case. In the following, we demonstrate the formalism by using one element of each index set. Without loss of generality, we choose ABL for inherent cases, ACC for dependent cases, and NOM for unmarked cases. For *Lex* the lexicon of G:

(16) a. $compat_{\textsc{abl}} = \{x \in D : label(x) = l, l \in Lex, \texttt{N} \in features(l),$
$exponent(l)$ is ablative$\}$
b. $compat_{\textsc{acc}} = \{x \in D : label(x) = l, l \in Lex, \texttt{N} \in features(l),$
$exponent(l)$ is accusative$\}$
c. $compat_{\textsc{nom}} = \{x \in D : label(x) = l, l \in Lex, \texttt{N} \in features(l),$
$exponent(l)$ is nominative$\}$

If a leaf in t has an exponent that is syncretic between multiple cases, this leaf will appear in multiple sets.[6] For each leaf, we will define a shortcut predicate stating the "actual" (syntactic) case it has, out of all the cases it is morphologically compatible with. We will call these predicates $inh_{\textsc{abl}}$, $dep_{\textsc{acc}}$, etc. This determination depends on the leaf's context: is it selected for by an LI that requires a certain case? Does it stand in a dependent case licensing configuration? What is its case assignment domain?, and so on. In order to define these "actual" case predicates, we first introduce some auxiliary predicates. These predicates hold of a tuple of nodes iff the formula on the right-hand side of $\Leftrightarrow_{\text{def}}$ evaluates to $\mathbf{t}$.

A node x c-commands a node y iff x does not strictly dominate y (this excludes $x \equiv y$), and all nodes z that strictly dominate x also stricly dominate y (this excludes y dominating x):

[6] For linguistic arguments in support of this approach, see the feature indeterminacy problems discussed in [8–10].

(17) $ccom(x, y) \Leftrightarrow_{\text{def}} \neg x \trianglelefteq y \wedge (\forall z : z \triangleleft x \rightarrow z \triangleleft y)$

A node x is a phase head iff its category feature is either `C` or `Voice` or `D`. (Other definitions are possible depending on one's theory of phases.)

(18) $phase(x) \Leftrightarrow_{\text{def}} Feat_{\texttt{C}}(x) \vee Feat_{\texttt{Voice}}(x) \vee Feat_{\texttt{D}}(x)$

A node x is in case assignment domain i if there is a phase head y with category feature i that c-commands x and there is no phase head z distinct from y that c-commands x but does not also c-command y.

(19) $$domain_i(x) \Leftrightarrow_{\text{def}} (\exists y : phase(y) \wedge Feat_i(y) \wedge ccom(Sliceroot(y), x) \wedge (\forall z : (phase(z) \wedge ccom(Sliceroot(z), x)) \rightarrow (z \equiv y \vee ccom(Sliceroot(z), y))))$$

Two nodes x and y are in the same case assignment domain iff all phase heads z that c-command one also c-command the other:

(20) $$samedomain(x, y) \Leftrightarrow_{\text{def}} (\forall z : phase(z) \rightarrow (ccom(Sliceroot(z), x) \leftrightarrow ccom(Sliceroot(z), y)))$$

We are now ready to define the when a leaf has a particular actual inherent case: A nominal leaf x with potential inherent case c has c as its actual case iff x is merged by an LI y that selects for c.

(21) For k the maximum number of positive features on LIs in the range of *label*:
$$inh_{\text{ABL}}(x) \Leftrightarrow_{\text{def}} compat_{\text{ABL}}{}^{\uparrow}(x) \wedge (\exists m : m \blacktriangleleft Sliceroot(x) \wedge (\exists n : \textstyle\bigvee_{1 \leq i \leq k}(Proj_i(n) \equiv m \wedge assigns_{\text{ABL},i}(n))))$$

Since each inh_c for some case c is a pseudonym for a 1-place predicate $D \rightarrow \{\mathbf{t}, \mathbf{f}\} \in P^1$, we can define a set of these predicates, (22). This then provides us with the second-order predicate "$inh^{\uparrow}$".

(22) $inh = \{inh_i : i \in \textsc{Inh}\}$

A potential dependent-down case is an actual case iff the nominal x has the required exponent and does not carry inherent case and is c-commanded by another nominal y in the same case assignment domain that does not have an actual inherent case. The case assignment domain is specified for each case, here `C` for ACC.

(23) $$dep_{\text{ACC}}(x) \Leftrightarrow_{\text{def}} compat_{\text{ACC}}{}^{\uparrow}(x) \wedge domain_{\texttt{C}}(y) \wedge (\forall p : inh^{\uparrow}(p) \rightarrow \neg p(x)) \wedge (\exists y : samedomain(x, y) \wedge (\forall p : inh^{\uparrow}(p) \rightarrow \neg p(y)) \wedge ccom(Sliceroot(y), x))$$

Note that dep_{ACC} contains the second-order predicate $inh^{\uparrow}$. Strictly speaking, this is not necessary. We could have expressed $(\forall p : inh^{\uparrow}(p) \rightarrow \neg p(x))$ as the conjunction $\neg inh_{\text{ABL}}(x) \wedge \neg inh_{\text{LOC}}(x) \wedge \neg inh_{\text{DAT}}(x)$, iff the natural language we

model contains exactly these inherent cases. As we will see shortly, this holds for all second-order shortcut predicates. Thus the present characterization of Dependent Case Theory is in fact first-order definable.[7] However, we will continue the second-order notation for readability purposes.

For a dependent-up case, replace $ccom(Sliceroot(y), x)$ with $ccom(Sliceroot(x), y)$.

We define the set of all shorthand predicates denoting actual dependent case:

(24) $dep = \{dep_i : i \in \textsc{Dep}\}$

Potential unmarked cases are licensed as actual cases if the structural conditions for dependent cases are not met. DCT uses assignment rules where the assignment of accusative takes precedence over the assignment of nominative. Thus it is excluded that nominative could arise in a structural environment where accusative is attested. However, when employing a licensing view to DCT, we not only need to make sure that accusative is licensed only if there is a c-commander, but also that nominative is licensed only if there is no c-commander. (For domains with a dependent-down case like CP in ergative languages: absolutive is licensed iff it doesn't c-command another eligible NP.[8] For domains with both dependent-up and dependent-down cases: both conditions must hold.) We thus need to treat unmarked cases as Baker [2] negative dependent cases, although for a different reason. For the formalism, this does not matter: we remain inside MSO-definability.

(25) $$\begin{aligned} unm_{\textsc{nom}}(x) \Leftrightarrow_{\text{def}} & \; compat_{\textsc{nom}}{}^{\uparrow}(x) \wedge domain_{\textsc{c}}(x) \wedge \\ & (\forall p : (inh^{\uparrow}(p) \vee dep^{\uparrow}(p)) \rightarrow \neg p(x)) \wedge \\ & \neg(\exists y : samedomain(y, x) \wedge ccom(Sliceroot(y), x) \wedge \\ & \quad (\exists q : (dep^{\uparrow}(q) \vee unm^{\uparrow}(q)) \wedge q(y))) \end{aligned}$$

(26) $unm = \{unm_i : i \in \textsc{Unm}\}$

A potential default case becomes an actual case if the nominal's potential inherent, dependent, and unmarked cases are all not its actual case.

(27) $$\begin{aligned} def(x) \Leftrightarrow_{\text{def}} & \; compat_{\textsc{def}}{}^{\uparrow}(x) \wedge \\ & (\forall p : (inh^{\uparrow}(p) \vee dep^{\uparrow}(p) \vee unm^{\uparrow}(p)) \rightarrow \neg p(x)) \end{aligned}$$

We now have predicates determining for each node n whether n is a nominal LI carrying a certain syntactic case. The predicates are defined in such a way that out of $inh \cap dep \cap unm \cap \{def\}$, only one of these predicates will be $\mathbf{t}$ of n.

[7] This is a welcome result. MSO can capture many patterns that we do not expect to find in the case distribution in natural language. For instance, MSO-logic can implement modulo-counting, i.e., for a sequence of NPs, the case of the NP depends on its position in the sequence modulo some integer.

[8] This follows the traditional formalization of DCT [23]. More recently, data have shown that ERG–ERG–ABS patterns are unattested [2, 24] and realized as ERG–ABS–ABS instead. This paper does not aim to contribute to this issue.

3.3 Defining the MSO-constraint

We are now ready to formulate a version of the Case Filter that assures that every nominal will have a syntactic case. We state the following MSO-constraint on derivation trees generated by our MG:

(28) $\forall x : Feat_{\mathbb{N}}(x) \rightarrow (\exists p : (inh^{\uparrow}(p) \vee dep^{\uparrow}(p) \vee unm^{\uparrow}(p) \vee \{def\}^{\uparrow}(p)) \wedge p(x))$

We walk through some scenarios to get an impression of a completeness and soundness proof for (28) with respect to the algorithm in fig. 1. (28) is complete iff every syntactic tree whose cases have been assigned by fig. 1 fulfills (28). (28) is sound iff every tree that fulfills (28) will be assigned the same cases by fig. 1.

The algorithm assigns exactly one case to any NP. It doesn't assign more than one case because it exits the algorithm as soon as a case is assigned. It doesn't assign no case at all because of its catch-all assignment of default case. Similarly, (28) doesn't license more than one syntactic case on any NP due to the definitions of dependent, unmarked, and default cases ensuring that no case higher on the case assignment hierarchy in (1) is licensed as well.

It is easy to see that for intransitives, the algorithm will assign an unmarked case corresponding to the domain the single NP is in, and (28) will only license this unmarked case on the single NP. For regular transitives, the algorithm will assign a dependent case to one of the two NPs, and an unmarked case to the other. (28) will license exactly these cases as well. For transitives that assign an inherent case to one argument α, the algorithm will assign this inherent case to α and unmarked case to the other argument (for languages of the Icelandic-type). (28) will only license this assignment.

Incorrect configurations include dependent case on the single NP argument of an intransitive. The algorithm will not assign this. (28) will not license this. Another incorrect configuration is a regular transitive with both arguments marked with the same type of dependent case (both up or both down). The algorithm will not assign this. (28) will not license this. Finally, the algorithm will not license unmarked case on two NPs standing in a c-command relation. (28) will also not license this, due to the explicit condition on unmarked cases that there must not be a c-commanding/c-commanded NP in the same case assignment domain.

3.4 Conclusions

Graf [16] has shown that any constraint that can be defined in MSO logic can be added to an MG that is strongly equivalent to an MG without this constraint. In this section, we saw that the distribution of morphological case as defined by DCT (section 1.2) can be formalized as MSO constraint. Thus it follows that the rules of DCT, including one-to-many licensing in the assignment of dependent-down cases, is not more complex than other constraints commonly imposed onto MGs.

We now turn to a new question: What is the relationship between the class of constraints that have an MSO-formalization, to the class of constraints captured

by Move-dependencies? Are they equivalent, like MSO-constraints and Merge-dependencies, or is one more expressive than the other? Schematically,

(29) Merge $\Leftrightarrow$ MSO $^{?}\Leftrightarrow^{?}$ Move

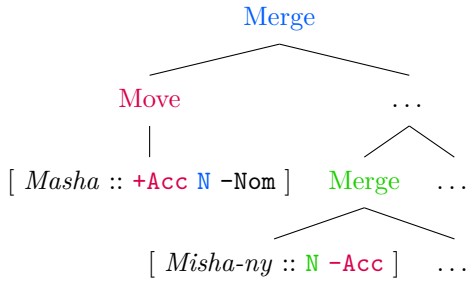
We take a step towards exploring this question in the next section by attempting to formalize the rules of Dependent Case Theory as Move-dependencies.

4 Dependent-down case as Move is non-regular

Alternatively to specifying the distribution of cases via MSO-statements about the configurations these cases can appear in, we treat Case as a feature on lexical items with a compatible exponent and "check" it against a licensor. This checking can happen at a distance and corresponds to the deletion of "uninterpretable" features under Agree in early Minimalist literature; [6], et seq., i.a. We explore the option of taking cases to be licensee features `-f` on the nominal that carries case.

For dependent-down case, we take the c-commanding nominal as licensor. In a clause with a transitive verb, accusative case is assigned to the object in the presence of the c-commanding subject. This is schematized in (30).

(30)



4.1 Generalizing the movement type

Our definition of the distribution of Move-nodes and the application of the operation *move* in def. 2 does not work for (30). Applying *move* to the feature sequence on the child of Move, ⟨`+Acc N -Nom`⟩, is undefined. This problem arises because the only kind of movement defined in sec. 2 was raising movement: the Move node must dominate the LI x carrying the negative feature; thereby the LI y projecting this Move node must have selected for the subtree containing x. However, in (30), this is not the case. The projecting LI *Masha* does not select for anything; rather, it connects to the subtree containing *Misha-ny* by being selected.

This mismatch arises because the movement in (30) is not raising. Graf [15] has shown that MGs can handle other (generalized) movement types besides raising, including lowering and sideward movement, different sizes of the moved constituent (head, phrase, pied-piped phrase), and different linearizations (covert,

to the left, to the right). In order to define these movement types, he recognizes that the requirement that a Move-node m can be associated to a feature `-f` on an LI l only if m dominates l, is a special case of an MSO-definable relation between m and l.

(31) $R^{\text{RAISING}}(m, l)$ iff $m \trianglelefteq l$

Other movement types may be defined in terms of different relations. The movement type DD employed by dependent-down case licensing uses the relation R^{DD} between a Move-node m associated with feature `-f` on a lexical item l, (32), and the movement type DU for dependent-up case licensing the relation R^{DU} in (33):

(32) $R^{\text{DD}}(m, l)$ iff $ccom(Sliceroot(m), l)$

(33) $R^{\text{DU}}(m, l)$ iff $ccom(Sliceroot(l), m)$

Like with raising, it holds that every `-f` must have a Move-node as occurrence, and that every Move-node can only be an occurrence of one negative `-f` feature. Also like raising, we impose a minimality requirement: the Move-node that is an occurrence of a feature `-f` must be the closest possible Move-node in terms of R^{DD}. Additionally, DD movement is bounded by case assignment domains. Both the Move-node and the LI with `-f` must be in the same domain, and this domain must be of a particular type.

After [15], we define a constraint on which Move-nodes can be associated to which `-Acc` features: For a non-branching node l labeled with `-Acc` and a Move-node m, m is an occurrence of l iff they stand in relation R^{DD} and are in the same case assignment domain which is a CP, and there is no other non-branching node x with either `-Acc` or `+Acc` that is closer in terms of R^{DD}:

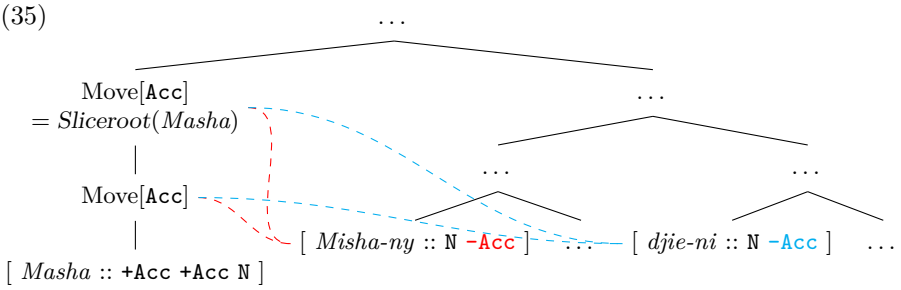
$$\begin{aligned}(34)\quad \forall m : \forall l : (&Label_{\text{Move}}(m) \wedge Feat_{\texttt{-Acc}}(l)) \rightarrow \\ &(Occ(m, l) \leftrightarrow (R^{\text{DD}}(m, l) \wedge domain_{\texttt{C}}(m) \wedge samedomain(m, l) \wedge \\ &\qquad (\forall x : (Feat_{\texttt{-Acc}}(x) \vee Feat_{\texttt{+Acc}}(x)) \rightarrow \\ &\qquad\quad (R^{\text{DD}}(x, l) \rightarrow R^{\text{DD}}(x, m)))))\end{aligned}$$

4.2 Generating non-regular tree languages

Recall the two configurations for dependent-down case assignment discussed in sec. 1.2. In the 1:n-configuration, a single nominal n licenses case on multiple accusative-marked lexical items. In our system above, this corresponds to n projecting multiple Move-nodes, which each stand in a R^{DD} relation with one of the case-marked LIs.

However, (34) does not define which Move-node is an occurrence of which `-Acc`. This ambiguity is illustrated in (35).

(35)



This is not a problem per se as nothing in the grammar depends on which Move-node associates with which LI. I will argue that any derivation will succeed as long as there exists a bijection.

However, this implies that the grammar allows a single licensor to enter into arbitrarily many licensing relationships. In this section, we have silently been assuming that the lexicon contains multiple versions of every non-inherently case-marked nominal: one with `+Acc` and one without. However, if a nominal can carry arbitrarily many `+Acc` features, the lexicon will grow from finite to infinite. Since the non-terminals in our tree grammar are made up of feature sequences on lexical items, the grammar will no longer be regular. This constitutes a significant increase in complexity of our formalism, which we would like to avoid in the absence good reason to adopt it. The proof is straight-forward, for instance with the pumping lemma for regular tree languages [7, §1.2]. Section 5 will propose an amendment to the formalism that allows us to remain in the realm of regularity.

4.3 Conclusions

In this section, we have seen that an MG that formalizes dependent-down case as **lic**-features checked by the c-commanding nominal does not generate regular derivation tree languages. This is due to there being no restriction on the number of instances of dependent-down case that a single nominal can license, and to the requirement in our grammar of a one-to-one relationship between Move-nodes projected by `+Acc` features and `-Acc` features (the SMC). Regular languages cannot count to arbitrarily large numbers.

However, in sec. 3 that in general a tree language with an equivalent distribution of accusative-marked nominals as L *is* regular and expressible by MGs by refining its **sel**-features via MSO-constraints. It follows that a formalization with case as **lic**-features is not equivalent to a formalization with case as **sel**-features.

In the next section, we will explore a possibility to use **lic**-features while not increasing the complexity of the derived tree language beyond regular tree languages. We will achieve this by suspending the SMC for selected features.

5 Selectively suspending the SMC

The formalization in sec. 4 increases the complexity of MGs in two ways. First, for dependent-down case licensing in 1:n-configurations, the licensor carries an `+Acc`

feature for every licensing relationship. This results in unboundedly large lexical items and thereby an infinite lexicon. The second problem is the requirement that the number of `+Acc` features must match the number of Move-nodes projected by these `+Acc` features, and also the number of `-Acc` features on the licensees resp. the number of licensees. This requires our tree grammar to count, which is not possible for a regular tree grammar. This stands in clear contrast to the results of sec. 3.

5.1 Persistent licensor features

We can dissociate the number of `+Acc` features on the licensor from the number of projected Move-nodes by allowing a single `+Acc` to project multiple Move-nodes. Taking inspiration from Stabler's [31] persistent licensee features, we allow the licensor feature `+Acc` to be persistent. We add the rule in (37) in addition to the existing *move*-rule from def. 1, repeated in (36):

$$\frac{\langle \texttt{+Acc} f_2 \cdots f_k, \phi_1, \ldots, \phi_{i-1}, \texttt{-Acc} g_2 \cdots g_l, \phi_{i+1}, \ldots \phi_m \rangle}{\langle \texttt{+Acc} f_2 \cdots f_k, \phi_1, \ldots, \phi_{i-1}, g_2 \cdots g_l, \phi_{i+1}, \ldots \phi_m \rangle}\ move_2 \tag{36}$$

We also amend the definition of MGs from def. 4 to contain the production rule in (37):

$$A_i \rightarrow \text{Move}(A_j) \qquad \text{if } A_i = move_2(A_j) \tag{37}$$

This will restrict the necessary lexical items to 2 per noun.[9] We have now solved the first of our two problems: the number of `+Acc` features no longer needs to match the number of projected Move-nodes and of `-Acc` features in the derivation. However, the second problem remains: the number of Move-nodes needs to match the number of `-Acc` features. In sec. 5.2, I propose a solution to both problems by keeping the number of `-Acc` features finite.

[9] We could even reduce this to a single lexical item per noun. This is irrelevant for the formalism because the blow-up to the lexicon is constant and thereby negligible, but it does have linguistic significance. Given that the phonological content of a potential licensor virtually never changes with respect to whether it is a licensor or not (J. Bobaljik, p.c.), we would not like our formalism to employ accidental homophony to capture this because doing so would allow us to derive non-homophonous pairs of licensors, thereby overgenerating with respect to the linguistic data. We have two options to capture this systematic syncretism. The first one is to assume a generative lexicon that contains entries of the one type (either licensors or not) and derives entries of the second type. The second option is to introduce a rule that deletes `+Acc` from a nominal and applies optionally, (i), and to add the scheme for production rules in (ii) to our MG.

(i) $$\frac{\langle \texttt{+Acc}\ f_2 \cdots f_i\ \texttt{N}\ f_j \cdots f_k, \phi_1, \ldots, \phi_n \rangle}{\langle f_2 \cdots f_i\ \texttt{N}\ f_j \cdots f_k, \phi_1, \ldots, \phi_n \rangle}\ del$$

(ii) $A_i \rightarrow A_j$ if $A_i = del(A_j)$; $A_i, A_j \in N$.

5.2 Suspending the SMC

The solution lies in abandoning the requirement that each Move-node can only associate with one `-Acc` feature. If a single Move-node could associate with unboundedly many `-Acc` features, we would lose the one-to-one requirement that cannot be expressed by regular tree grammars.

This requirement is known as the SMC and can be paraphrased as "Every Move-node can be associated with one negative feature." What I propose is to change this definition to: "Every Move-node that is not projected by `+Acc` is associated with exactly one negative feature." The association of one Move-node to arbitrarily many `-Acc` features allows the formalism to not distinguish between the exact number of `-Acc` features and Move-nodes, but to only differentiate between two states: "at least one `-Acc` feature needs checking" and "no `-Acc` feature needs checking". We alter the functions *merge* and *move* defined above to implement this. `-Acc` features are deleted when they do not contribute to the distinction between these two states, i.e., when there already is another `-Acc` awaiting association with a Move-node.

We define the function $merge_2$ that reduces two `-Acc` to one. For $\mathtt{x} \in \mathbf{sel}$, $f_{j_1}, g_{j_2}, h_{j_3}, i_{j_4} \in \mathit{Feat}$, $0 \leq j_1 \leq k$, $0 \leq j_2 \leq l$, $0 \leq j_3 \leq m$, $0 \leq j_4 \leq n$; and $\Phi, X, \Psi, \Omega \in (\mathit{Feat}^*)^*$:

$$(38) \qquad \frac{\langle \mathtt{=x} f_2 \cdots f_k, \Phi, \mathtt{-Acc}\ g_2 \cdots g_l, X \rangle \qquad \langle \mathtt{x} h_2 \cdots h_m, \Psi, \mathtt{-Acc}\ i_2 \cdots i_n, \Omega \rangle}{\langle f_2 \cdots f_k, \Phi, \mathtt{-Acc}\ g_2 \cdots g_l, X, h_2 \cdots h_m, \Psi, i_2 \cdots i_n, \Omega \rangle}$$

We alter the definition of *merge* in def. 1 on p. 10, repeated as (39), to apply subordinately to (38), i.e., only if it is not the case that for some $i \leq l, j \leq n$, ϕ_i and ψ_j start with `-Acc`:

$$(39) \qquad \frac{\langle \mathtt{=x} f_2 \cdots f_k, \phi_1, \ldots, \phi_l \rangle \qquad \langle \mathtt{x} g_2 \cdots g_m, \psi_1, \ldots, \psi_n \rangle}{\langle f_2 \cdots f_k, \phi_1, \ldots, \phi_l, g_2 \cdots g_m, \psi_1, \ldots, \psi_n \rangle}\ merge_1$$

We define the function $move_3$ as in (40) to avoid introduction of a second initial `-Acc` feature. For $\mathtt{x}, f_{j_1}, g_{j_2}, h_{j_3} \in \mathit{Feat}$; $0 \leq j_1 \leq k$, $0 \leq j_2 \leq l$, $0 \leq j_3 \leq m$; $\Phi, X, \Psi \in (\mathit{Feat}^*)^*$; and `-x -Acc` $g_3 \cdots g_l$ preceding or following `-Acc` $h_2 \cdots h_m$:

$$(40) \qquad \frac{\langle \mathtt{+x} f_2 \cdots f_k, \Phi, \mathtt{-x\ -Acc}\ g_3 \cdots g_l, X, \mathtt{-Acc}\ h_2 \cdots h_m, \Psi \rangle}{\langle f_2 \cdots f_k, \Phi, g_3 \cdots g_l, X, \mathtt{-Acc}\ h_2 \cdots h_m, \Psi \rangle}\ move_3$$

The rule *move* from def. 2 on p. 10, repeated in (41), applies only when it is not the case that g_2=`-Acc` and for some $j, 1 \leq j \leq n$, the first element in ϕ_j is `-Acc`:

$$(41) \qquad \frac{\langle \mathtt{+x} f_2 \cdots f_k, \phi_1, \ldots, \phi_{i-1}, \mathtt{-x} g_2 \cdots g_l, \phi_{i+1}, \ldots, \phi_n \rangle}{\langle f_2 \cdots f_k, \phi_1, \ldots, \phi_{i-1}, g_2 \cdots g_l, \phi_{i+1}, \ldots \phi_n \rangle}\ move_1$$

A Minimalist Grammar licensing dependent-down case with licensor features without increasing the complexity of its derivation tree language beyond regularity, is then defined as follows:

Definition 4 (MG, licensing dependent-down case). *An MG* $G = \langle N, T, S, P\rangle$ *is a regular tree grammar with* $T = Lex \cup \{\text{Merge}, \text{Move}\}$. *The non-terminals* N *are tuples of sequences of features occuring in Lex,* $N \subset (Feat^+)^+$, *namely* $N = \{\langle s_0, \ldots, s_{|\mathbf{lic}|}\rangle : s_i \in \{\beta : \alpha\beta = features(l), l \in Lex\}, 0 \leq i \leq |\mathbf{lic}|\}$, *i.e., all* $(|\mathbf{lic}|+1)$*-tuples made up of feature sequences that occur as suffix of the feature sequence of some lexical item. The start symbol* $S = \langle \mathtt{C}, \varepsilon, \ldots, \varepsilon\rangle \in N$, $\mathtt{C} \in F$. *The production rules* P *are defined based on the feature calculus operations* $merge_1, merge_2, move_1, move_3$ *as defined in (39), (38), (41), and (40), respectively. For* $A_i, A_j, A_k \in N$*:*

1. $A_i \to \text{Merge}(A_j, A_k)$ *if* $A_i = merge_1(A_j, A_k)$ *or* $A_i = merge_2(A_j, A_k)$
2. $A_i \to \text{Move}(A_j)$ *if* $A_i = move_1(A_j)$ *or* $A_i = move_3(A_j)$
3. $A_i \to l$ *if* $A_i = features(l)$, $l \in Lex$.

The new rules $merge_2$ and $move_3$ do not increase the weak nor the strong generative capacity of MGs. The same string language and phrase structure languages are derivable by MGs with MSO-definable constraints, as demonstrated in sec. 3. This is so because the present movement is not actual movement: no phonological or semantic features are displaced. It serves merely the feature calculus operations determining the distribution of case morphology. The aim of this paper has been to show that this can be achieved with refined **sel**-features as well as with SMC-liberated **lic**-features.

6 Conclusions

I have shown that dependent-down cases pose a problem for MGs because they allow one-to-many licensing relationships involving one licensor and arbitrarily many licensees. MGs with a one-to-one association between licensor nodes and licensee features (i.e., with the SMC) would need to generate non-regular derivation tree languages in order to ensure that the number of licensor nodes and licensee features match.

However, the high-level rules for dependent-down case assignment are well within the range of regular tree languages. I have shown that this is the case by providing a monadic second-order formula, which is known to be imposable onto an MG by refining its Merge-features [16]. We can thus see an asymmetry between the operations Merge and Move: while these constraints can be expressed as refined Merge-features, they cannot be expressed via Move-features while staying in the same complexity class.

I have argued that it is possible to realize dependent-down case as Move-features if the SMC is suspended. This amounts to adding new rules for *merge* and *move* that will discard a newly activated `-Acc` feature if there is already an active `-Acc` in the derivation. In this way, we can formalize a single licensor node being able to license arbitrarily many licensee features.

We have motivation to abandon the SMC for case licensing because it affects neither word order at the PF interface nor scope at the LF interface. However, this does not extend to other instances of one-to-many licensing dependencies

in natural languages: quantifier raising and multiple wh-fronting affect LF and PF respectively and so cannot be licensed in this way. Note that Gärtner and Michaelis's [12, 13] approach of wh-clustering requires the wh-elements to stand in pairwise c-command relationships. This is parallel to the "daisy-chain" configuration of dependent-down case licensing, and does not raise an issue in either construction. The present paper makes an empirical prediction about multiple wh-fronting: On the assumptions in [12, 13] about wh-fronting, for constructions in which not all fronted wh-words stand in a c-command relation to another fronted wh-word (in their respective base positions), the PF-order of moved wh-phrases will be either completely free (arbitrary) or completely fixed (determined by post-syntactic linearization rules), but is crucially not determined in the derivation.

I leave an empirical investigation of this phenomenon, as well as of QR and the attestation of unbounded 1:n dependent-down case assignment for future research.

Acknowledgments Many thanks to Jonathan Bobaljik, Thomas Graf, Tim Hunter, Stefan Kaufmann, Jos Tellings, and Susi Wurmbrand, as well as to three anonymous reviewers for Formal Grammar, and two anonymous reviewers for the NASSLLI Student Session for helpful feedback and stimulating discussions. All errors are my own.

References

1. Akkuş, F.: On Iranian case & agreement (2017), talk given at the University of Vienna, Austria, on December 15
2. Baker, M.: Case: Its principles and its parameters. Cambridge University Press (2015)
3. Baker, M., Bobaljik, J.: On inherent and dependent theories of ergative case. Oxford University Press (2017)
4. Baker, M., Vinokurova, N.: Two modalities of case assignment: case in Sakha. Natural Language & Linguistic Theory **28**(3), 593–642 (2010)
5. Bobaljik, J., Wurmbrand, S.: Questions With Declarative Syntax Tell Us What About Selection? In: Ángel Gallego, Ott, D. (eds.) 50 Years Later: Reflections on Chomsky's Aspects, MIT Working Papers in Linguistics, vol. 77. MIT Press (2015)
6. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge, MA (1995)
7. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (2008), release November 18, 2008
8. Dalrymple, M., Kaplan, R.M.: Feature Indeterminacy and Feature Resolution. Language **76**(4), 759–798 (2000)
9. Dalrymple, M., King, T.H., Sadler, L.: Indeterminacy by Underspecification. Journal of Linguistics **45**(1), 31–68 (2009)
10. Eisenberg, P.: A Note on "Identity of Constituents". Linguistic Inquiry **4**(3), 417–420 (1973)

11. Ermolaeva, M.: Morphological Agreement in Minimalist Grammars. In: Foret, A., Muskens, R., Pogodalla, S. (eds.) Formal Grammar. FG 2017. Lecture Notes in Computer Science (LNCS), vol. 10686, pp. 20–36. Springer, Berlin, Heidelberg (2018)
12. Gärtner, H.M., Michaelis, J.: On the treatment of multiple-wh-interrogatives in Minimalist Grammars. In: Hanneforth, T., Fanselow, G. (eds.) Language and Logos, pp. 339–366. Studia grammatica, Akademie Verlag, Berlin (2010)
13. Gärtner, H.M., Michaelis, J.: In Defense of Generalized Wh-Clustering. In: Baglini, R., Grinsell, T., Keane, X., Singerman, A., Thomas, J. (eds.) Proceedings of the 46th Annual Meeting of the Chicago Linguistic Society: The Main Session. pp. 137–146. The Chicago Linguistic Society, Chicago (2014)
14. Graf, T.: Closure Properties of Minimalist Derivation Tree Languages. In: Pogodalla and Prost [25], pp. 96–111. https://doi.org/10.1007/978-3-642-22221-4
15. Graf, T.: Movement-Generalized Minimalist Grammars. In: Béchet, D., Dikovsky, A.J. (eds.) Logical Aspects of Computational Linguistics. Lecture Notes in Computer Science, vol. 7351, pp. 58–73 (2012). https://doi.org/10.1007/978-3-642-31262-5
16. Graf, T.: Local and Transderivational Constraints in Syntax and Semantics. Ph.D. thesis, UCLA (2013), `http://thomasgraf.net/doc/papers/PhDThesis_RollingRelease.pdf`
17. Keenan, E., Moss, L.: Mathematical Structures in Languages. University of Chicago Press (2015)
18. Kobele, G., Retoré, C., Salvati, S.: An automata-theoretic approach to minimalism. In: Rogers, J., Kepser, S. (eds.) Model-Theoretic Syntax at 10, 13–17 August 2007, organized as part of ESSLLI 2007 (2007), `https://www.yumpu.com/en/document/view/34852154/`
19. Kobele, G.M.: Minimalist tree languages are closed under intersection with recognizable tree languages. In: Pogodalla and Prost [25], pp. 129–144. https://doi.org/10.1007/978-3-642-22221-4
20. Kornfilt, J., Preminger, O.: Nominative as 'no case at all': an argument from raising-to-ACC in Sakha. In: Joseph, A., Predolac, E. (eds.) Proceedings of the 9th Workshop on Altaic Formal Linguistics (WAFL9). MIT Working Papers in Linguistics, vol. 76, pp. 109–120. MIT Press (2015)
21. Levin, T.: Successive-cyclic case assignment: Korean nominative-nominative case-stacking. Natural Language & Linguistic Theory **35**, 447–498 (2017). https://doi.org/10.1007/s11049-016-9342-z
22. Maling, J.: Of Nominative and Accusative: The Hierarchical Assignment of Grammatical Case in Finnish. In: Holmberg, A., Nikanne, U. (eds.) Case and Other Functional Categories in Finnish Syntax. Mouton de Gruyter (1993)
23. Marantz, A.: Case and licensing. In: Westphal, G., Ao, B., Chae, H.R. (eds.) Proceedings of the Eastern States Conference on Linguistics (ESCOL) 1991. pp. 234–253. Cornell University, CLC Publications, Ithaca, NY (1992)
24. Nie, Y.: Why is there NOM-NOM and ACC-ACC but no ERG-ERG? In: Lamont, A., Tetzloff, K. (eds.) Proceedings of NELS 47. vol. 2, pp. 315–328 (2017)
25. Pogodalla, S., Prost, J.P. (eds.): Lecture Notes in Artificial Intelligence, vol. 6736. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22221-4
26. Poole, E.: A Configurational Account of Finnish Case. University of Pennsylvania Working Papers in Linguistics **21**, Article 26 (2015), `https://repository.upenn.edu/pwpl/vol21/iss1/26`

27. Poole, E.: The locality of dependent case (2016), http://ethanpoole.com/handouts/2016/poole-dependent-case-locality.pdf, presented at GLOW 39 (April 5) and WCCFL 34 (April 30; no proceedings paper)
28. Preminger, O.: Agreement and its Failures, Linguistic Inquiry Monographs, vol. 68. MIT Press (2014)
29. Salvati, S.: Minimalist Grammars in the Light of Logic. In: Pogodalla, S., Quatrini, M., Retoré, C. (eds.) Logic and Grammar. Essays dedicated to Alain Lecomte on the occasion of his 60th birthday, Lecture Notes in Computer Science, vol. 6700. Springer, Berlin, Heidelberg (2011)
30. Stabler, E.: Derivational minimalism. In: Retoré, C. (ed.) Logical Aspects of Computational Linguistics. First International Conference, LACL '96, Nancy, France, September 23-25, 1996. Selected Papers, Lecture Notes in Artificial Intelligence, vol. 1328, pp. 68–95. Springer (1997). https://doi.org/10.1007/BFb0052147
31. Stabler, E.: Computational Perspectives on Minimalism. In: Boeckx, C. (ed.) The Oxford Handbook of Linguistic Minimalism, chap. 27, pp. 617–642. Oxford University Press (2011). https://doi.org/10.1093/oxfordhb/9780199549368.001.0001
32. Stabler, E., Keenan, E.: Structural similarity within and among languages. Theoretical Computer Science **293**(2), 345–363 (2003)
33. Vinokurova, N.: Lexical categories and argument structure: a study with reference to Sakha. Ph.D. thesis, University of Utrecht (2005)
34. Yip, M., Maling, J., Jackendoff, R.: Case in tiers. Language **63**(2), 217–250 (1987)