2

Towards a Minimalist Machine

Sandiway Fong and Jason Ginsburg

2.1 Introduction

With respect to computational modeling, the Minimalist Program (MP) (e.g. Chomsky 1995b; 2001 and thereafter), has not lent itself to straightforward formal characterization. MP models have undergone a series of methodological and theoretical shifts in the continual search for a better characterization of the language faculty. Although computational efficiency is of central concern in cognitive models, a so-called "third factor" principle, this does not imply that theoretical improvements will necessarily make for more efficient implementation, as will be discussed in this chapter.[1] However, it is important to recognize the computational implications of these shifts, evaluate the challenges, and adopt appropriate computational architectures. This chapter describes an efficient architecture that has been implemented and tested on a wider variety of syntactic phenomena than any comparable system that we know of. We discuss in detail the reasons for, and consequences of, our architectural choices.

2.2 Free or feature-driven movement

Let us begin with a concrete example of a theoretical shift: movement, or Internal Merge (IM) in MP terms, has been alternately characterized as being "driven" (e.g. Chomsky 1995b; 2001) or "free" (e.g. Chomsky 2013; 2015b). By "driven", we simply mean that preconditions are placed on movement. The EPP, or Edge-feature (Chomsky 2001), is an example of a mechanism that can drive and limit displacement. A general requirement for grammatical feature-matching during IM can also serve the same purpose. In these theories, no movement may take place without featural approval. The Minimalist Grammar (MG) formalism (Stabler 1997; and much work since) codifies this feature-driven approach. Note that explicit feature checking is not the only way to condition displacement; for example, the Minimal Link Condition (MLC) and the Last Resort rule (Chomsky 1995b) are economy-based conditions that do not involve

[1] According to (Chomsky 2005), there are three factors that enter into the growth of the language for the individual: (i) genetic endowment, (ii) experience, and (iii) principles not specific to the faculty of language.

Minimalist Parsing. First edition. Robert C. Berwick and Edward P. Stabler (eds.)

(explicit) feature checking[2] The Antilocality proposal of Bošković (2016) is yet another. Pesetsky and Torrego's T to C movement is an interesting case of economy that we will discuss in detail in this chapter because it involves a combination of feature checking and operation counting. By "free" we simply mean that there are no preconditions on IM; i.e. by default, constituents are free to displace anywhere without checking first for permission. Within this framework, a theory that can dispense with EPP/Edge and other non-substantive features (perhaps invented solely for the purpose of constraining displacement) carries less "technical baggage" to burden acquisition or biological endowment. Obviously, since not all possible derivations are admissible; structural constraints that filter out uninterpretable or ill-formed syntactic objects (SOs) must appear somewhere "downstream" from Merge. Ideally, these independently- motivated constraints would originate from language-external factors, e.g. the need to reformat SOs to meet the requirements for exchanging relevant information with external systems, or from limits on general cognitive architecture. For example, at the Conceptual–Intensional (CI) interface, Merged objects must be explicitly labeled (Chomsky 2013; 2015b). Thus, Labeling limits possible phrasal movement. Third factor considerations could also limit the range and depth of structures computed[3]

Let us turn our attention to computational issues for the two types of theories outlined above. First, a feature-driven model, if sufficiently constrained, can result in an "efficient" implementation in the sense that SOs can be derived with few, or no, choice points. Suppose the possible syntactic operations are limited to the following three: (i) External Set Merge (ESM), (ii) Internal Set Merge (ISM), (iii) External Pair Merge (EPM)[4] In the model described here, a machine state (SO, LIs) will consist of a current SO and an ordered list of (unmerged) lexical items (LIs)[5] ESM will merge the first LI *H* with the current SO α, forming {*H*, α}. More succinctly, we can write (α, [H, ..]) ↦ ({H, α} [..]) (We use the notation [H, ..] to represent a list (or stack) with first (or top) element H. [..] represents the remainder of the list once H is removed.) ISM identifies a sub-constituent β in α, forming {β, α}; i.e. (α, [H, ..]) ↦ ({β, α}, [H, ..]). (We will write β ⊂ α to indicate that β is a proper sub-constituent of α.) In principle, β could be any subconstituent of α; however, we use featural identification and locality to limit our choice to just a single β. For example, the interrogative complementizer C_Q will seek a β[*wh*], where *wh* is a feature. We may limit ISM to the closest available β[*wh*][6] Finally,

[2] The decision procedure *P* (described later) contains an implementation of the Last Resort rule.

[3] For example, without constraints on IM, we can select any sub-SO for displacement, so the combinatorics of SO growth becomes a problem. One third-factor hypothesis that can be investigated is that cognitive computation not only does not support, but actively suppresses, the creation of infinite loops. Further exploration of this issue falls outside the scope of this chapter.

[4] We put aside a possible fourth type of operation, Internal Pair Merge (IPM). See Richards (2009) and Epstein, Kitahara, and Seely (2016).

[5] Our implementation assumes an Oracle that turns a set of LIs into a properly ordered list. This convenient simplification is purely for efficiency's sake.

[6] Of course, by "closest" we mean the nearest structural constituent as defined by the c-command domain, rather than the nearest compatible constituent in terms of linear order.

EPM will form <*H*, α>, with H as adjunct.[7] Using the notation introduced above, this is (α, [H,..]) ↦ (<H, α> [..]). These are the choices available to the machine. (We put aside for now the situation that obtains when H is not a simple head, but instead a complex SO formed earlier in a lower workspace (WS) and placed at the front of the LIs.)

Suppose the machine has a decision procedure P that deterministically selects the right operation in order for the derivation to converge. If P exists, by always making the right choice, all other candidate operations can be pruned immediately. This is the optimal situation from the point of computational minimalization, in terms both of memory demand and of the number of operations required to converge on the desired SO. Should P make a wrong choice, the derivation crashes, i.e. fails to converge on a single SO. In the case where we intend a crash, e.g. to show a SO is underivable from some particular initial list of heads (LIs), the machine containing P need not show that there is no combination of available operations to circumvent failure; it simply needs to fail to advance its derivation at some point (perhaps as early as possible). An empirically adequate P will produce convergent derivations without backtracking when started with appropriate LIs, and crash when not. Ideally, P will be able to also cite a grammatically relevant reason for a crash, not just simply halt.

The Oracle-like knowledge encoded in P will be a combination of well-motivated linguistic principles, e.g. minimal search, plus (irreducible) ad hoc stipulations.[8] Let us define a simple, or "minimal," P as one that decides on the right operation solely on the basis of the current state. A more powerful P, e.g. one that employs lookahead (by precomputing future machine states) or considers prior machine states, e.g. in Markovian fashion, to decide on the right move, will be deemed architecturally non-optimal. We describe such a minimal P in the following sections.

A free Merge model cannot entertain the level of determinism described above for feature-based systems. If Merge is unrestricted, the operations ESM, ISM, and EPM are active and available at all relevant stages of the computation. Irrelevant operations must be blocked somewhere downstream. For example, consider a transitive verb construction {T, {SBJ, {v*, {V, OBJ}}}}, where the subject SBJ is first Merged at the edge of v*P. In typical feature-based theories, tense T has an EPP (or Edge) feature. Assuming minimal search, this triggers ISM to raise SBJ to the surface subject position (in preference to the object OBJ). In the free Merge framework, there is no EPP feature; therefore both {SBJ, {T, {~~SBJ~~, {v*, {V, OBJ}}}}} (raising) and {T, {SBJ, {v*, {V, OBJ}}}} (no raising) will be put forth to be Merged with the complementizer C.[9] We must rely on the Labeling algorithm at Transfer to the CI interface to rule out the illicit English structure

[7] Convention: in this chapter, in the ordered pair <α, β>, the first element α always represents the adjunct, and β the head.

[8] We speculate (without experimental evidence) that the ad hoc component potentially could be acquired. In terms of the decision procedure P, currently, we manually sort the available operations: ESM, ISM, and IPM. (Learning to more efficiently assemble SOs through experience would be the driving principle.) If so, an individual actually has no ad hoc *P* to begin with. Instead, it might be created through error-driven exposure to data.

[9] Notation: strikethrough will be used throughout this chapter to indicate displacement.

{C, {T, {SBJ, {v*, {V, OBJ}}}}}, as in Chomsky (2013; 2015b)[10] This non-determinism results in combinatorial explosion that minimal P avoids by design[11]

2.3 Computational cyclicity

Cyclicity also has important consequences for implementation. First, the notion of incrementally assembling SOs in strict bottom-up fashion has not always enjoyed unqualified support in the MP. For example, although it seems to violate the No-Tampering Condition (NTC) of Chomsky (2008), the notion of syntactic head-to-head movement (after phrasal movement) reappears from time to time. For example, Chomsky (2015b) indicates that the root R should raise in {v*, {R, OBJ}} and form the amalgam [R-v*], realized as a reverse-ordered Pair-Merged head-head structure. In short, from {v*, {R, OBJ}}, we non-cyclically create {<v*,R>, {R,OBJ}}[12] Simultaneous (or parallel) IM operating at the Phase level, as in (Chomsky 2008), is another proposal that seems to violate the idea of the strict cycle at the single-Merge level of granularity. Let us discuss each of these in turn with implementation of minimal P in mind.

Commitment to a strict version of the cycle results in the simplest possible scheme for scheduling operations. For example, suppose a head gets just one (and only one) opportunity, at the time of its first Merge to the current SO, to probe, value, and have its features valued. Once a head is submerged into a larger SO, the system "forgets" and never resurrects the "now buried" head for further probing duties. More precisely, given machine state (SO, [H ..]), Set Merge of H and SO proceeds by first having (the features of) H probe its intended c-command domain SO, and possibly carry out operations such as Agree(H, β), where β ⊂ SO. (Recall that SO is the current syntactic object and H is the first in the list of LIs.) If successful, the new machine state will be ({H, SO}, [..]). We consider H "buried", and the machine never digs into the current SO to have buried heads (re-)probe. (Note this does not prevent H from subsequently being a goal targeted by operations like Agree.) The proposed scheme is the simplest possible because ~~otherwise would~~ require the system to organize and maintain a list of delayed-past-Merge probe-goal operations. Bookkeeping of this nature requires additional memory and computational resources, so we do not consider it for our minimal P. In fact, whenever we have opportunities to bookkeep, we will pass on

[10] An attempt could be made to make a minimal P-type device available in a free Merge model. However, it is not clear whether this would be technically feasible (or deducible from assumptions) in all possible scenarios or computational states. Even in our small example above, to forestall speculative (non-)movement, this would involve building in the knowledge that {T, {SBJ, {v*, {V, OBJ}}} cannot be labeled unless ISM of something that shares identical ϕ-features with T happens next (but not ESM or EPM). We must also deduce that the possibility of ESM is ruled out because ϕ-features in an external constituent cannot satisfy the identity requirement. Finally, EPM must be ruled out on the grounds that the canonical ϕ-feature sharing configuration would not be achieved.

[11] Note that in free Merge, we find echoes of Move-α, a concept in the earlier Principles-and-Parameters (P&P) framework (see Chomsky 1981). Move-α implies that syntactic objects of any category may, in principle, undergo IM. However, the EPP is very much part of the P&P framework; there is no EPP in free Merge.

[12] Obviously, the timing of head-to-head movement is critical if the operation is to meet the conditions of strict cyclicity. See also Epstein et al. (2016) and Collins and Stabler (2016) for relevant discussion.

them. (However, as will be explained later, this does not prevent the implementation of Multiple Agree (Chomsky 2001).)

Consider example (1a): as discussed in Chomsky (2008), both phrases $[..]_1$ and $[..]_2$ must simultaneously raise from the post-verbal object position in (1b) in order to sidestep a possible violation of the subject island condition. (Compare (1a) with (1e).)

(1) a. Of which car was the driver awarded a prize = (7ii) (Chomsky 2008)

b. [[*of which car*]$_1$ was [*the driver of* ~~*which car*~~]$_2$ awarded [~~*the driver* [*of which car*]$_1$~~]$_2$ a prize]

c. {C_Q {T, {.. {award, *the driver of which car*}}}}

d. {[*the driver of which car*]$_2$, {T, .. {award, [~~*the driver of which car*~~]$_2$ a prize}}

e. *Of which car did the driver cause a scandal = (6ii) (Chomsky 2008)

If we take the pre-IM structure to be (1c), it should be apparent that the raising of *the driver of which car* to the Edge of T is counter-cyclic. Note that the naïve version, i.e. raising to the Edge of T first, and forming (1d) before merging interrogative C_Q, does not work. Merge of C_Q will trigger probing, but the closest *of which car* is inside the subject island. In our implementation, a stack data structure originally introduced for computational efficiency saves the analysis.

The machine stack K operates as follows:

(2) (i) In machine state (α[!F], [H, ..]), where !F denotes an unvalued feature F and α[!F] denotes a SO α with unvalued feature F, α[!F] is pushed onto K if α is no longer the current SO, i.e. stacking is triggered when something is merged with α[!F].

(ii) When a probe requires a matching goal, it must be found on K, i.e. only the stack is consulted.

Given (2), if we apply ESM to machine state (α[!F], [H, ..]), we obtain a new machine state ({H, α[!F]}, [..]), and unvalued feature !F is buried inside the SO ({H, α[!F]}. To avoid a possibly deep (and expensive) search of the current SO each time a probe is first Merged, we simply look to the stack for candidate-matching goals. In a probe-goal model, because the stack (by definition) only contains constituents with unvalued features, we can be confident we have all the candidates of interest without examining the current SO. Finally, for maximal computational efficiency, i.e. eliminating SO search, we can demand that the top element of the stack always contains the right goal.[13]

Returning to example (1a), *which car was the driver awarded a prize*, the DP *the driver of which car* lacks Case when it is first Merged as object of the verb root *award*,

[13] Probes search for a particular unvalued feature. If the stack top has cached the right element, there is no search, and the data structure proposed is maximally efficient. If it does not, the next element down in the stack can be tested, and so on. A "cache miss" is non-optimal, but a convergent derivation can still be obtained. In informal testing on our corpus, the first stack element is nearly always the one we want.

creating SO {award, DP}. Since the DP is no longer the current SO, it goes on the stack. Proceeding in strict cyclic fashion, we obtain {T, {v, {.. {award, DP}}}}. In our grammar, T has an Edge feature, and hence the DP raises to form {DP, {T, {v, {.. {award, ~~DP~~}}}}}. Next, C_Q merges and probes the SO; but as we have outlined above, it actually grabs the DP initially left on the stack, and the derivation proceeds.[14] To be clear, the stack we have just introduced is not a conceptually necessary component of the architecture. The notion of a machine stack is not normally part of the formal vocabulary of the linguist; therefore, it requires some additional justification. We have introduced it as a computational efficiency measure, largely to sidestep goal search. It happens also to facilitate parallel Merge of the sort discussed in Chomsky (2008).

At this point, a remark on maintaining computational cyclicity in the face of paradigmatic shift is in order. Grammatical operations need not always operate in "lockstep" with the core Merge implementation as sketched above. Although Merge is the driving force in our implementation—i.e. operations such as Agree and probe-goal are triggered immediately at first Merge of a head—this "first wave" approach may not be desirable, or even necessary, from the point of view of empirical coverage or efficient implementation. An example is Labeling of the sort discussed in Bošković (2016), where he exploits differences in the timing of Labeling for various phrases to account for a variety of effects.[15] Computationally speaking, choices exist here; for example, we can choose to always label a phrase as soon as possible, e.g. as in Ginsburg (2016), or delay labeling for some phrases, as in Bošković (2016). Immediate Labeling is Merge-driven and respects the strict cycle. Delayed Labeling, e.g. until Transfer as proposed in Chomsky (2015b), can be viewed as non-cyclic, as Labeling is triggered only on completion of a Phase. Determining the label of deeply buried phrases long after Merge seems, at first glance, to be computationally inefficient. This is true for largely deterministic models, but in the case of free Merge, delaying Labeling may actually be a computational win, simply because large numbers of candidate SOs (introduced non-deterministically) will be ruled out early anyway. In short, why bother to label for Transfer if most candidate SOs won't make it there?

In the system described here, we do not implement algorithmic Labeling; instead, P deterministically assign a label to each syntactic object at Merge-time. Updating to a loosely coupled labeling algorithm poses a challenge to our model of quickly-evaluate-and-forget. One way to reconcile the two paradigms for future implementation is to place Labeling in a "second wave" of cyclic operation. Under this proposal, Labeling proceeds cyclically but exhibits variable latency; i.e. it is simply delayed, and not forced

[14] For (1e), **of which car did the driver cause a scandal*, there is a Phase boundary (v*P) separating C from the initial Merge of *the driver of which car* to the Edge of v*. Since Phase boundaries are pushed onto the stack in our implementation, we could make use of this to rule out (1e). However, the actual implementation makes use of an i-within-i-style stacking constraint: "if $[_A$ .. $[_B$ ..]] is pushed onto the stack, and $[_B$ ..] from a substream is already on the stack, A subsumes B and renders B unavailable on the stack." This means the stack element for *the driver of which car* blocks the required stack element (*of*) *which car*, and thus (1e) cannot be formed.

[15] By "not in lockstep" we mean that SOs need not be Merged and labeled in complete synchronicity; in particular, the label for a SO can be determined post-Merge (cf. Label Accessibility Condition (LAC): Chomsky 2000).

to make a decision about the label of the result of every Merge. These and other shifts pose concrete challenges to settling on a particular theory to implement, never mind settling on a particular formal or mathematical framework such as minimalist grammars (Stabler 1997).

2.4 Architecture overview

Let us review the components of the implementation introduced so far: we have defined a machine state as consisting of a current syntactic object (SO) and an ordered list of heads (LIs). We have also argued earlier for an unvalued feature (!F) stack. Therefore, the components of the machine state are now as given in (3):

(3) (SO, !F Stack, LIs)

Operations, to be defined below, map machine states to machine states. Let us note that operations have severely limited access to each of the components identified above for computational efficiency reasons as follows:

A. The current syntactic object (SO) is opaque except for access to top-level grammatical features. We assume that features percolate upwards according to the headedness of constituents of the SO. In other words, the features of {α, β} will be determined by P, selecting recursively from the features of α (or β). In more contemporary terms, our model integrates Labeling tightly with Merge P, with P deciding on the label of {α, β} immediately.[16]

B. The unvalued feature stack (!F Stack) is accessed for all probe-goal operations. No access is made to the current SO. The first matching stack element is retrieved. However, Phase boundaries are also registered on the stack. Since access cannot proceed beyond a Phase boundary, and stack order equals hierarchical distance, the Phase Impenetrability Condition (PIC) (Chomsky 2001) and the MLC are largely accounted for.[17]

C. As discussed earlier, we assume the list of heads (LIs) is pre-ordered for convergent computation. Operations can only access the first element of the list; i.e. there can be no operation that selects e.g. the 3rd, LI to be merged with the current SO.

2.5 Fundamental operations

Operations are defined below in terms of machine state. First, let us formally define the fundamental Merge operations (introduced earlier):

[16] Deterministic immediate Labeling will not always be available, e.g. Set Merge of XP and YP creates {XP, YP}. P will be forced to create a choice point producing both [$_{XP}$ XP, YP] and [$_{YP}$ XP YP].

[17] These MP conditions largely supersede earlier P&P framework concepts such as Government, Subjacency, Barriers (Chomsky 1986), and Relativized Minimality (Rizzi 1990).

(4) a. External Set Merge (ESM):
(α, K, [H, ..]) ↦ ({α, H}, K', [..])
stack: K' = K, i.e. unchanged, unless α[!F]; if so push α onto K to create K'

b. Internal Set Merge (ISM):
(α, K, [..]) ↦ ({βα}, K, [..]), β ∈ K

c. External Pair Merge (EPM): (α, K, [H, ..]) ↦ (<H,α>, K, [..])

(Recall that machine states, given in (3), are triples of the form (SO, K, LIs). Operations define mapping (↦) of states to states. Mathematically speaking, operations need not be functions, as will be discussed later. In (4a), α refers to the SO, K refers to the !F Stack, and [H, ..] refers to the input stream of LIs with H as the first element.)

ISM in (4b) selects a sub-SO β of α from the stack K. This means that there must (originally) be some unvalued feature F accessible in β[!F]. ISM is thus a feature-driven operation with one exception. Suppose a feature !F is valued on β, then the question arises whether we should delete it from the stack. It may come as a surprise to the reader that we do not remove anything from the stack. As a result, stack items remain available for further (Merge) operations. To search for and remove inactive stack elements is a bookkeeping task that incurs a computational penalty. We always skip these chores. Also, by not deleting valued stack elements, it is possible to implement movement such as topicalization and rightwards displacement without resorting to the invention of non-substantive unvalued features for the sole purpose of keeping a stack element "active" (necessary in a strict feature-checking model).

We also model Leftwards Thematization/Extraction (TH/EX) (Sobin 2014), making use of already stacked DPs[18] In the expletive construction shown in (5a), the object *a book* raises leftwards over *taken,* the verbal past participle (cf. (5b)). In our grammar, an invisible light verb ~~v~~ (following Sobin) attracts an already stacked sentential object (with unvalued Case), as ~~v~~ possesses an Edge feature. In the corresponding active sentence, (5c), there is no ~~v~~, and *a book* does not raise.[19]

(5) a. There was *a book* taken = (27c) (Sobin 2014)
b. *There was taken *a book* = (27b) ibid.
c. John took *a book*

A second group of fundamental operations involve the shifting of Workspaces (WS). Associated with a WS is a current SO plus stack and input list, and a single stream of operations (determined by P) that empties the input list and converges on a single SO. Consider the sentence (6a) below. There are two DPs that must be independently

[18] It can be argued that stylistic phenomena such as TH/EX should not be within the purview of narrow syntax: see Chomsky (2001). It is not clear to us how this can be accomplished at the PF interface, as it seems to involve the displacement of entire DPs. Hence, we have chosen to model it as part of syntactic movement, following Sobin (2014).

[19] Another way to implement apparent leftwards movement would be to add features to move the verb around instead of the object DP. However, in our model this would require potentially stacking all verbs.

assembled. In our model, we pre-load the input list as shown in (6b). (Note that there are two distinct *uncomplicated* LIs shown: these are not copies.)

(6) *a.* An uncomplicated man makes uncomplicated plans
b. [*plans*, *uncomplicated*, *make*, v*, [*man*, *uncomplicated*, *an*], T, C]

In the current model, sub-LIs may be pre-inserted at strategic locations within the main input list. Sub-LIs induce WS management operations. A sub-LI specifies both an independent stream of LIs to be assembled and also the position of the computed sub-SO in the main stream. In the example above, [*man*, *uncomplicated*, *an*] is placed at the position where we want the DP to be inserted back into the main stream after sub-stream computation completes. Since it is a sentential subject, we specify it should be inserted between the verbal categorizer v* and tense T. The machine proceeds from left to right, and recursively merges *plans*, *uncomplicated*, *make*, and v* in one uninterrupted sequence of operations to form the intermediate SO {v*, {*make*, {*uncomplicated*, *plans*}}}. The corresponding machine state is shown below in (7a).[20]

(7) a. ({v*, {*make*, {*uncomplicated*, *plans*}}}, _, [[*man*, *uncomplicated*, *an*], T, C])
b. (man, _, [*uncomplicated*, *man*])
c. ({*an*, {*uncomplicated*, *man*}}, _, [])
d. ({v*, {*make*, {*uncomplicated*, *plans*}}}, _, [{*an*, {*uncomplicated*, *man*}}, T, C])
e. {{*an*, {*uncomplicated*, *man*}}, {v*,{*make*, {*uncomplicated*, *plans*}}}}, _, [T, C])
f. ({C, {DP, {T, {DP, {v*, {*make*, {*uncomplicated*, *plans*}}}}}}}, _, [])

In state (7a), the machine encounters the sub-LI [*man*, *uncomplicated*, *an*] next. This signals the machine to enter a sub-WS and begin a new sub-stream computation. The sub-WS has *man* as the first SO and remaining LIs [*uncomplicated*, *an*], as in (7b). This computation ends when a single SO {*an*, {*uncomplicated*, *man*}}, a DP, is formed and there are no more LIs left ([]), as in (7c). The machine then terminates the sub-stream, and pops back up to the saved main-stream machine state, but with the newly computed DP inserted at the front of the LIs, as given in ~~(3d).~~ ESM of the SO with DP = {*an*, {*uncomplicated*, *man*}} creates {DP, {v*, {*make*, {*uncomplicated*, *plans*}}}}. The derivation then converges with {C, {DP, {T, {DP, {v*, {*make*, {*uncomplicated*, *plans*}}}}}}}, as shown in ~~(3f).~~

More generally, we can define the two WS shift operations shown in (8a,b):

(8) a. Down WS (DWS): (α, K, [[H, ..], ..]) ↦ (H, [], [..])
b. Up WS (UWS): (β, _, []) ↦ (α, K, [β, ..])

In (8a), the first element of the input itself is an input stream ([H, ..]), and H becomes the new initial SO of the sub-WS. Note that α, K, and the rest of the input stream are temporarily forgotten, and restored only in (8b). β is transferred from the sub-WS to be the first element of the higher input stream. (8a,b) represent a serial view

[20] Notation: an underscore placeholder is used for the stack to indicate that its value is not relevant to this discussion.

of computation implied by the presence of sub-LIs. In a highly deterministic model such as the one described here, a serial implementation is a reasonable choice. In a free Merge model, where non-determinism dominates the computational landscape, separately forming all sub-SOs associated with sub-LIs first, and then substituting the completed SOs back into the LIs before beginning main-stream computation will always be the more computationally efficient choice.[21]

2.6 The decision procedure P

The decision procedure P is at the heart of our model. In fact, one can argue it is actually the "brains" of the model, in the sense it decides which operation should issue given any particular machine state. As mentioned in an earlier section, correct decision-making requires oracle-like powers. We have aimed for a minimal P that is as deterministic as it is practical. The system does compute multiple SOs in some cases—we return to consider an example of this in a later section; but by and large, it is locally deterministic, issuing one fundamental operation (defined in the above section) per machine state. Further, we claim our machine produces correct derivations for a wide variety of examples from the literature. (For space reasons, we cannot list the examples or their derivations. However, we can demonstrate the veracity of our claim[22]) The question then arises as to how we came up with this particular decision procedure P. The short answer to this important question is simply by inspection (and trial and error) as we incrementally introduced examples to the system. This is also the reason why we stated earlier that P has an ad hoc component (see also note 9).

The following groups of actions are presented in strict order and constitute ad hoc P; i.e. only the first matching action will be triggered. As a result, ad hoc P is largely deterministic. (Cases where P is non-deterministic will also be discussed.) Each action makes use of the following template:

(9) (SO, Stack, LIs) ↦ (SO', Stack', LIs')
[*Preconditions on* SO, Stack, *and* LIs]
stack: [Stack' *derived from* Stack]
[*Labeling for constructed* SO']

(Recall that machine states have signature (SO, Stack, LIs).) We construct machine state (SO', Stack', LIs') from (SO, Stack, LIs) if stated preconditions are met. The last two lines will specify stack operations and Labeling for the new current SO, respectively. For each action, we will briefly list an example that motivates its inclusion; however, the reader

[21] The reason is that the results of sub-LI computation can be shared (and not repeated) within separate threads of execution in the case of non-determinism. This way memorization is achieved for free, but discussion of the details and its effects would take us beyond the scope of this chapter.

[22] Full step-by-step derivations are provided for all the examples that we cover in http://elmo.sbs.arizona.edu/sandiway/mpp/binding-examples/.

is referred to Ginsburg and Fong (Chapter 3 this volume) for derivation walkthroughs and further details.

We begin with the only actions that do not follow the template introduced in (9):

(10) Phase Completion: (α, [β,..], [..]) ↦ (α, K', [..])
if phase(α), where α is current SO,
stack K' = [β, ..] with inactive(SOs) fronted

Basically, this states that completion of a Phase will result in stack visibility changes. First, some background: in the case of syntactic pronominal Binding, we employ a Kayne (2002)-style doubling constituent (DC).[23] In the case of a sentence like (11a), we begin its derivation with the LIs listed in (11b).

(11) a. John_1 thinks (that) he_1 is smart
b. [john, d, he, d, smart, v_{be}, T, c_e, think, v_{unerg}, T, c]
c. {smart, {d!case, {he, {d!case, John}}}}
d. [$_{CP}$ (that) he_1 is smart]
e. *John_1 praises him_1

In (11b), the prefix [*john*, d, *he*, d, ...] produces the DC {d, {*he*, {d, *john*}}}. This is a DP with an internal DP; we will refer to the entire DP as a DC. At the interface, DC [$_d x$ [$_d$ *y*]] will mean that *x* and *y* are coreferential. (We assume noun phrases are DPs; however, we will sometimes abbreviate a DP using just the head noun.) Since the r-expr *John* is in a subsidiary position within the DC, it will have to undergo IM to a theta-position, and get its Case feature valued.

Since the DC contains unvalued Case, it will be stacked when the adjective *smart* is merged in (11c), in accordance with (2i). (We assume the locus of unvalued Case (!case) to be the D head.) Although *John* is buried inside the DC, since it also has unvalued Case, it must also be stacked. However, we stipulate that the subsidiary DP must be stacked with status "inactive", meaning it is rendered inaccessible to search. Activation of inactive stack elements falls to the PC actions in (10).

Upon completion of a Phase, e.g. (11d), (10) fronts previously inactive *John* in the stack, and activates it. As a result, {d!case, John} becomes visible to IM, and !case to probe-goal. *John* then takes up the matrix subject position. Note that the requirement of activation at a phase boundary is necessary in order to block (11e). If *John* is available upon initial stacking, (11e) would be predicted to be grammatical.

(12) Halt: (α, [..], [])

(12) is the simplest action: if there are no more LIs ([]), the machine halts, and SO α is the result. If α contains any unvalued features, the machine crashes.[24]

The next mini-group of actions concerns merger of a determiner (d) from the LIs:

[23] For further details about modeling of Kayne (2002), see Fong and Ginsburg (2012a).

[24] Although our machine halts in this state, in theory additional IM operations may be possible, e.g. topicalization. We do not explore this option.

(13) Action Group: D Merge

i. (n, K, [d ..]) ↦ ({n, d}, K', [..])
in d[!n] and n[!d], !n and !d are valued
stack: K' = push n and/or d onto K if n or d have unvalued features
d labels {n, d}

ii. (d_1, K, [d_2 ..]) ↦ ({d_1 d_2}, K, [..])
d's share !case and !n
$d_1$labels { d_1 d_2}

External Set Merge of *d* from LIs to the SO *n* in (13i) is accompanied by mutual uninterpretable feature valuation for !*n* and !*d*. This seems redundant: i.e. why not simply merge *d* and *n* without checking off !*n* and !*d*? The answer is: in our grammar, some determiners, e.g. d_{rel}, used for relative clauses, do not value uninterpretable !*d*. We derive (14a) from the LIs in (14b). The object DP {d_{rel}, book!d} will contain *book* with unvalued *d* (viz. *book*!d). After constructing the clause headed by c_{rel}, *book*!d raises to form (14c). Since *book*!d is a head, it also labels (14c).[25]

(14) a. the book (that) I read
b. [book, d_{rel}, read, v*, [i, d], T_{past}, c_{rel}, the]
c. {book!d, c_{rel}P} (c_{rel}P derives "(that) I read")
d. {the, {book, c_{rel}P}}

In the final step for the construction of the relative clause, the determiner *the*, a head that is generally able to value !*d*, merges with (14c), creating (14d). The new head *the* values !d on *book*. This completes our discussion of action (13i).

Action (13ii) was created for the case of exclamatives, such as in (15a):

(15) a. What a silly book!
b. {what, {a, {silly, book}}}

In our grammar, both *what* and *a* are analyzed as determiners. Since Case on the DP in (15a) will be valued just once, we have *what* and *a* sharing unvalued features (!case and !n).

(16) Doubling Constituent (DC) Merge:
(α, K, [β ..]) ↦ ({α, β}, K' , [..])
if β is a DC [$_d$ x γ]
stack: K' = K + β[!F_1] + inactive γ[!F_2]
SO α labels {α, β}

Earlier, we introduced Kayne's notion of a DC for pronominal Binding. Action (16) will stack both the entire DC (β) and the subsidiary DP(γ) (inactivated) from a lower

[25] In a more elaborate theory, *book* in (13) would actually be composed of a root *book* + categorizer *n*, a functional head. A root R would not be able to label {R, XP}. Another strategy will be needed. The details are beyond the scope of this chapter.

WS. Both β and γ have unvalued Case; γ also has an unvalued interpretable θ-feature. See earlier discussion of sentence (11a).

(17) Action Group: Merge to Edge of T
i. (T[!Edge], K, [EXPL,..]) ↦ ({T, EXPL}, K', [..])
first(LIs) is EXPL, e.g. *there*
stack: K' = K + β if β[!F], otherwise K' = K
SO T labels {T, EXPL}
ii. T[!Edge], K, [..]) ↦ ({T, β}, K, [..])
β ∈ K
SO T labels {T, β}

In action group (17), we find a potential conflict between the two actions listed: however, ~~(16i)~~ precedes and blocks ~~(16ii)~~. In more detail, ~~(16i)~~, i.e. ESM of an expletive to the edge of tense T, is preferred over ~~(16ii)~~, i.e. ISM of an argument from the stack. In either case, T[!Edge] means that T must have an unvalued uninterpretable Edge feature (!Edge) for the action to fire[26]

Consider the expletive construction in (18a) and the machine state in (18b):

(18) a. There is a train arriving
b. ({T, ..{*arrive*, {*a*, *train*}}}, _, [*there* C])
c. A train is arriving

At machine state (18b), the correct move is to merge expletive *there* from the LI input, rather than to raise the DP {*a,train*}. Hence, (17i) must come before (17ii). Note that (17ii), not (17i), must be triggered in the derivation of ~~(17c)~~. (The crucial difference is that expletive *there* will not be present at the LI input in the case of (17c).)

(19) Theta Merge:
(XP[!θ], K, [Root,..]) ↦ ({Root, XP}, K', [..])
if Root is V (or A)
Root values !θ on XP
stack: K' = K + XP if XP[!F], otherwise K' = K
Root labels {Root, XP}[27]

In action (19), we find the basic case of a verbal (or adjectival) Root merging with an object. The Root assigns a θ-role to the object XP. In our grammar, potential arguments will have an unvalued interpretable θ-feature. Note that structural Case is not valued here. In Chomsky (2000; 2001), functional heads v* and T probe and value

[26] Edge of T is the classic EPP feature. After Merge obtains, the uninterpretable Edge feature is valued, and cannot be used again. This feature-driven approach is a weak point in our system: should the theory be updated to allow multiple merges to the edge of T, we would require multiple Edge features.

[27] In algorithmic Labeling, e.g. Chomsky (2013; 2015a), roots are too "weak" to label, and must be strengthened in the configuration {XP, {R, ~~XP~~}}, where XP and R share identical ϕ-features. Thus, Object Shift becomes obligatory. We do not model Object Shift in this system.

accusative and nominative Case, respectively.[28] If XP[!F] obtains, i.e. XP has some uninterpretable feature such as !Case, it is pushed onto stack K, and can be accessed by ISM or probe-goal.

(20) Internal Theta Merge to P:
(XP[!θ!case], K, [P, ..]) ↦ ({P, XP}, K', [..])
P values !θ and !case on XP
stack: K' = K + XP or K + {P, XP} if XP[!F], otherwise K' = K
P labels {P, XP}

Action (20) is very similar to action (19) (but see also action (22) below). In (20), a preposition P selects for an object (XP). The two significant differences are that: (i) P values uninterpretable Case for XP, and (ii) XP may be stacked[29] To accommodate pied-piping, either the object (XP) or the PP {P, XP} may be pushed onto the stack.[30] This non-determinism allows the generation of either (21a) or (21b) from the same list of LIs (21c):

(21) a. Of which car did they find the driver? (adapted from Chomsky 2008)
b. Which car did they find the driver of?
c. [car, which, of, driver, the, find, v*, [they, d], T_{past}, c_Q]

(22) Action Group: External Theta Merge to P:
i. (PP[!Edge], K, [XP[!θ],..]) ↦ ({~~XP PP~~}, K', [..])
PP = {P, YP}
P values !case and !θ on XP
stack: K' = K + XP if XP[!F], otherwise K' = K
P labels {XP, PP}
ii. (PP[!Edge], K, [..]) ↦ ({XP, PP}, K, [..])
XP[!θ] ∈ K
P values !case and !θ on XP
P labels {XP, PP}

Action (22i) permits the merger of an external argument XP to a preposition that supports one. The PP, ~~abbreviating~~ {P, YP} in (22i), must have an unvalued Edge feature. P also values interpretable !θ on XP, as well as assigning Inherent Case. Dyadic theme/goal *to* is an example of such a preposition; see example (23a), with our analysis given in (23b). Note that XP, from the LIs in (22i), is not a head, and therefore must

[28] Except, in later theories, e.g. Chomsky (2008; 2013), v* simply transmits its inflectional features to the verbal root. The verbal root then is responsible for valuing accusative Case. This has architectural consequences. In an earlier section, we stated that, for simplicity and efficiency, a head gets only a single chance, i.e. at ESM time, to probe and value. But in {v*, .. {V, OBJ}}, V cannot probe and value until it receives the inflectional features from v*. But this does not happen until v* is ESM'd, a situation that could be ungenerously viewed as a NTC (or strict cycle) violation.

[29] In action (19), V does not directly assign Case; instead, the higher functional head v* does. In action (20), P directly assigns Inherent Case. But see also note 30.

[30] The actual code is a bit more complicated. In our grammar, some prepositions may be empty, i.e. have no spellout. Following Pesetsky (1995), such prepositions may be used to analyze double object constructions, e.g. I gave [$_P$ [$_{DP}$ Mary] [[$_P$][$_{DP}$ a book]]]. There is no choice point generated in these cases.

have been formed in an WS earlier. (It must then have been uplifted to the LIs using operation (8b).)

(23) a. John gave a book to Mary
b. {{d, John}, {v*, {give ,{{a, book}, {*to*, {d, Mary}}}}}}

Action (22ii) differs from (22i) in that it grabs the external argument XP off the stack. This is to permit the subsidiary DP in a Doubling Constituent (DC) to θ-Merge. Consider sentence (24a):

(24) a. I showed John$_1$ himself$_1$ in the mirror
b. [john, d, he, self, G, [mirror, the, in], show, v* [i, d], T_{past}, c]
c. {self!case, {he, {d!case, john}}}
d. {G, {self!case, {he, {d!case, john}}}}
e. {{d!case, john}, {G, {self!case, {he, {~~d~~, ~~john~~}}}}}

(24b) is the list of lexical heads needed. The prefix [john, d, he, self, ..] forms the DC given in (24c). In our grammar, we assume *himself* = *self* + *he*, and therefore [$_d$ *himself*, [$_d$ *john*] is the relevant DC.[31] Following Pesetsky (1995), G is a dyadic preposition with the DC as object. *John* does not yet have Case or its θ-feature valued; hence *John* will be stacked upon Merge of G.[32] Action (22ii) will trigger at stage (24d) to form (24e).

(25) External Theta Merge to D:
(DP[!Edge], K, [XP[!θ!case], ..]) ↦ ({XP, DP}, K', [..])
DP = {d YP}
D values !case and !θ on XP
stack: K' = K + XP if XP[!F], otherwise K' = K
D labels {XP, DP}

Action (25) is very similar to action (22i), the crucial difference being that we ESM to DP (instead of PP). Action (25) is triggered for possessives in English; e.g. example (26a) receives the simplified analysis given in (26b)

(26) a. his dog
b. {{d, he}, {'s, {d, dog}}}

In (26b), *'s* is treated as a dyadic determiner with internal subject *he* and object *dog*. We assume that *he* + *'s* = *his*.[33]

[31] See also discussion of example (11a).

[32] Given the discussion of example (11e), the reader might reasonably expect *John* to be stacked inactive, and therefore inaccessible to action (22ii). However, in our grammar, the reflexive *-self* is analyzed as the head of the DP *himself*, and it permits active stacking by being a Phase head (a lexical stipulation). On completion of the DP headed by *-self*, it activates any inactive stack items in its domain, in accordance with the Phase Completion (PC) actions defined in (10). Thus, examples like *John*$_1$ *praises himself*$_1$ and (24a) are ruled in.

[33] The analysis given in (26b) is the simplest case. We make use of a Doubling Constituent (DC) for pronominal *he* for examples like *John*$_1$ *likes his*$_1$ *dog*. In other words, *he+'s dog* should be {{d, {*he*, {d, *john*}}}, {'*s*, {d, *dog*}}}. See also discussion of example (11a).

(27) Action Group: External Theta Merge to v*

i. (vP[!Edge], K, [XP[!θ], ..]) ↦ ({XP, vP}, K', [..])
vP = {v*, YP}
stack: K' = K + XP if XP[!F], otherwise K' = K
v* labels {XP,vP}

ii. (vP[!Edge], K, [..]) ↦ ({XP, vP}, K, [..])
vP = {v*, YP}, XP[!θ] ∈ K
v* labels {XP,vP}

Actions (27i,ii) parallel (22i,ii). The essential difference is we θ-merge to the Edge of transitive verbalizer v* in (27). Since (27i) precedes (27ii), we have expressed a preference for external Merge over IM. The situation in (28b) represents the crucial case:

(28) a. John$_1$ knows Peter realizes that Mary likes him$_1$
b. {v*, {realize, .. {d, {he, {d, john}}}..}}
c. Peter knows John$_1$ realizes that Mary likes him$_1$

In the case of example (28a), *Peter* must be externally Merged at stage (28b). However, the DC analysis of pronominal Binding means that *John* is also a candidate for Merge. The DC is [$_d$ *he* [$_d$ *john*]], and *John* will be stacked as initially inactive (see earlier discussion of (11)). On completion of the Phase [$_{CP}$ that Mary likes him], *John* will be activated and compete with *Peter* (from the input stream) for the subject position of v*-*realize*. The correct decision in this case is to take *Peter* from the input stream (external Merge) over *John* from the stack (IM).

Comparing (28a) with (28c), it is clear that we must take *John* from the stack in (28c). However, we have only the appearance of a reversal of choice. For (28c), *Peter* will be further downstream in the LIs, and therefore unavailable at stage (28b). Our preference of Merge over Move can be maintained.[34]

(29) Merge v and VP:
(VP, K, [v, ..]) ↦ ({v, VP}, K', [..])
v probes K: Agree(v, G) if goal G ∈ K
stack: K' = K + v
v labels {v, VP}

Action (29) triggers probe-goal for categorizers that need to value Case, e.g. v*. We assume that all verbal roots (V) must have some categorizer v. However, not all categorizers need probe. For transitive sentences such as (30a), v* must probe and value Case on the object (see (30b)). However, for intransitives like (30c), v in (30d) does not probe.

(30) a. John likes Mary
b. {v*, {like, {d!case, mary}}}

[34] In the case of action group (22), we have the same preference encoded when merging to the subject position of P. The crucial example(s) for that group are: *John*$_1$ *gave Peter*$_2$ *his*$_{1/2}$ *book*.

c. John runs
d. {v, {run}}

In some situations, the v probe itself could be a goal downstream. Since all probe-goal operations consult the stack, we stack v. For our grammar, the probe-as-goal scenario occurs in cases of auxiliary verb movement to C, such as (31a). In our grammar, auxiliary *will* is a v, a light verb stacked on v*, as shown in (31b).

(31) a. What will Mary buy?
b. {will, {{d, Mary}, {v*, {buy, {d, what}}}}} = *will*P
c. {T, *will*P}
d. {will, {T, *will*P}}
e. {c_Q, {{d, Mary}, {will, {T, *will*P}}}}
f. {{d, what}, {T, {will, {c_Q, {{d, Mary}, {~~will~~, {~~T~~, *will*P}}}}}}}

Merge of T in (31c) results in T probing not only with the purpose of Agree(T, *Mary*) but also for a unvalued v-feature present on auxiliaries like *will*. We stipulate that when this situation obtains, T attracts v in the sense of Pesetsky and Torrego (2001; hereinafter abbreviated to P&T). Suppose we assume that when a head *x* attracts another head *y*, *y* must raise to the edge of *x*. In the case of (31a), *will* raises to the edge of T, forming (31d).[35] Next, Mary raises to the surface subject position at the edge of T. Then interrogative head c_Q is merged. c_Q probes and attracts *wh*-item {d, what}. However, adopting the T to C story of P&T, c_Q also has an unvalued uninterpretable T feature. In the case of (31a), unvalued T on c_Q can be valued by attracting T. Therefore T raises to the edge of c_Q. Pied-piping ensures that *will* also raises to the edge of c_Q.[36] Assuming Spellout operates by only pronouncing the highest copy, auxiliaries appear in C in interrogatives.[37]

(32) Merge T and vP:
(vP, K, [T,..]) ↦ ({T, vP}, K, [..])
T probes K: Agree(T, G) if goal G ∈ K
T labels {T, vP}

Action (32) encodes the T counterpart to v* of action (29). Some instances of T, e.g. T_{past} (present in many of the examples above), will probe and value Nominative Case on a matching goal from the stack. Others, such as non-finite T embedded in *Peter likes to eat*, will not.

(33) Merge C and TP:
(TP, K, [c, ..]) ↦ (CP, K, [..])
c_Q probes K: CP = {G_1 ..{G_n, {c_Q TP}}..} for some goal(s) G_i ∈ K
c_{rel} probes K: CP = {G, {c_{rel}, TP}} for some goal G ∈ K

[35] Note that *will* does not label {*will* ,{T,vP}}. It is attracted to the edge of T, so T still labels.
[36] We assume transitivity of attraction here: i.e. if *x* attracts *y*, and *y* attracts *z*, then *z* also raises to *x*
[37] "The simplest assumption is that the phonological component spells out elements that undergo no further displacement—the heads of chains—with no need for further specification" (Chomsky 2001).

c_e probes K: CP = {G, {c_e, TP}} for some goal G ∈ K
c does not probe: CP = {c, TP},
c labels CP

Action (33) encodes merge of the complementizer C to the current SO TP. Interrogative c_Q triggers *wh*-phrase probing. As described in detail for example (31), this may also trigger a cascade of concomitant IMs to the edge of c_Q. In our grammar, non-interrogative C is relatively inert; it has no Edge feature, and IM is not permitted.[38] (We will return to consider the cases of c_{rel} and c_e later.)

(34) Pair Merge:
(α, K, [β, ..]) ↦ (<β, α>, K, [..])
α and β both non-heads
α labels <β, α>

Action (34) is deliberately situated as nearly the last possible action of ad hoc P. If none of the previous actions match, and α and β both are phrases, Pair Merge is permitted to apply. As currently defined, the adjunct comes from the input stream side only. This means the order of LIs supplied is crucial.

(35) a. John read a book in the gym
b. [book, a, [gym, the, in], read, v*, [john, d], T, c]
c. ({a, book}, _, [{in, {the, gym}}, read, v*, [john, d], T, c])
d. <{in, {the, gym}}, {a, book}>
e. {read, <{in, {the, gym}}, {a, book}>} = VP
f. {c, {{d, john}, {T, {{d, john}, {v*, VP}}}}}
g. [gym, the, in, [book, a], read, v*, [john, d], T, c]
h. ({in, {the, gym}}, _, [{a, book}, read, v*, [john, d], T, c])
i. <{a, book}, {in, {the, gym}}>

Consider sentence (35a). (35b) is a possible input stream for (35a). After constructing the object *a book*, the machine builds *in the gym* in a lower WS, and injects the PP back into the main stream just in front of the verb *read*. This machine state in given in (35c). Action (34) applies, and (35d) is created. The verb root *read* is merged next, forming (35e), and the derivation can proceed to converge as (35f). With regard to probe-goal search and further Merge operations, (35d) is identical to {*a*, *book*}.[39] The adjunct *in the gym* is opaque to further inspection.[40]

[38] See also note 29. We do not implement syntactic feature transmission. In recent accounts, e.g. Chomsky (2008), inflectional features are transmitted from C to T, and from v* to verbal root R. R is the proxy for v* that initiates Agree. In our grammar, T in the lexicon comes with the necessary inflectional features, and v* carries out Agree directly. One disadvantage of our approach is that we need to list multiple T-s in the lexicon. On the other hand, we get to preserve our simple computational model of (first) Merge probe only, thus avoiding extra search and bookkeeping.

[39] The adjunct *in the gym* can be adjoined at a higher level, e.g. to the verb *read*, to obtain a different reading for sentence (35a).

[40] Note pair merge should crash if <β[!F], α> results. Since the adjunct is opaque to the machine, the unvalued feature F can never be valued. *β[!F] (unimplemented) should precondition (34).

If we swap *book* and *gym*, as in (35g), (35h) would be the equivalent machine state, producing (35i), with the adjunct *in the gym* incorrectly identified as the head of the pair-merged phrase.

(36) Relabeling:
(α, K, [d,..]) ↦ ({d,{n,α}}, K, [..])
head n ∈ K
in d[!n] and n[!d], !n and !d are valued
n labels {n,α}, and d labels {d,{n,α}}

Action (36) is designed solely to fire in the case of relative clauses, and is an example of an experimental construction-specific rule.[41] Consider example (37a):

(37) a. the book which I read
b. *the book which that I read
c. [book, $which_{rel}$, read, v^*, [i, d], T_{past}, c_{rel}, the]
d. { $which_{rel}$, book!d}
e. {c_{rel} {{d, i}, {T_{past}, {{~~d,~~ ~~i~~}, {v^*, {read, {$which_{rel}$, book!d}}}}}}}
f. {{$which_{rel}$, book!d}, {c_{rel} {{d, i}, {T_{past}, {{~~d,~~ ~~i~~}, {v^*, {read, {~~$which_{rel}$~~, ~~book~~}}}}}}}}
g. {the, {book, {{$which_{rel}$, ~~book~~} c_{rel}P}}}

In (37c), the object DP *which book* will be relativized. To trigger this, we preselect $which_{rel}$, instead of regular determiner *which*, for the input stream. The determiner $which_{rel}$ has some special properties: (i) unlike regular *which*, it cannot check unvalued *d* on the noun (see (37d)); and (ii) it can check unvalued T, which will become useful at the sentential level.[42] When (37d) is formed, *book* has an unvalued feature, and so it is stacked. We proceed to build the clause headed by c_{rel}, shown in (37e). The head c_{rel} possesses both unvalued T and Rel features, and probes into the TP complement domain. IM of *which book* will satisfy both unvalued T and Rel on c_{rel} (due to the special properties of $which_{rel}$). (37f), = c_{rel}P, is formed. At this point, action (36) kicks in: it first raises previously stacked *book*!d to head {*book*!d, c_{rel}P}; this is the relabeling step. Then *the*, the last item on the input stream, is merged to form (37g).

Let us now consider (37b): why is this blocked? Following P&T (2001), the complementizer *that* is the spellout of T to C movement. T to C movement arises when a C, with unvalued T, attracts T. We have mentioned that c_{rel} possesses both unvalued T and Rel features; so c_{rel} could value these features by first attracting T to C (to value T), and then {$which_{rel}$ book} (to value Rel).[43] However, derivational economy blocks

[41] This action should be expanded and replaced by a more general rule governing IM of heads. Relabeling is currently only permitted for nouns for the sole purpose of forming relative clauses. Even in a tightly constrained feature-checking system, such as the one described here, the action should be generalized (with respect to constructions and categories).

[42] Relativization was also discussed earlier with respect to action (13).

[43] For dialects of English that do permit (37b), we must: (a) turn off economy, and (b) make sure *which book* is raised last to block **the book that which I read*. If there are dialects that allow all three versions, we need not worry about the order of raising.

this option in favor of the single operation of IM of {which$_{rel}$ book} to simultaneously satisfy both features. Hence (37b) is ruled out.

To summarize, we have presented a list of structure-building actions that all follow the template in (9). We have attempted to justify the inclusion of each one from theory and example.[44] Taking a step back, the listed actions can also be viewed as just "ground" instances, in terms of particular lexical and functional categories, of the fundamental operations listed in (4) and (8). For actions where we are reasonably sure the preconditions are deducible from theory, we can make a case that (that part of) P is not ad hoc. However, the order of the actions presented involves manual curation, and thus is wholly ad hoc.

2.7 Non-determinism revisited

We have already seen an example of non-deterministic behavior for cases of pied-piping such as in (21a,b), implemented using action (19). We have also seen above how Pesetsky and Torrego's economy condition can be extended to eliminate competing derivations in relative clause formation. Turning to examples (38a,b) we will now describe how P&T's condition can also lead to productive non-determinism in action (33).

(38) a. Mary thinks that Sue will buy the book
b. Mary thinks Sue will buy the book
c. [book, the, buy, v*, Sue, will, T, c_e, think, v_{unerg}, Mary, T, c]
d. { c_e, {Sue, {T, {~~Sue~~, {v*, {buy, {the, book}}}}}}}
e. {Sue, { c_e, {~~Sue~~, {T, {~~Sue~~, {v*, {buy, {the, book}}}}}}}}
f. {T, { c_e, { Sue, {T, {~~Sue~~, {v*, {buy, {the, book}}}}}}}}

Both (38a) and (38b) have the same starting input stream, (38c), in the system described here.[45] We first form the embedded clause, (38d), headed by c_e, using the prefix [*book*, *the*, *buy*, v*, *Sue*, *will*, T, c_e, ..]. We assume the complementizer c_e has unvalued T (and an Edge) feature. Hence, c_e must probe, and action (33) is engaged. There are two ways to value T in P&T's framework: (i) by attracting a Nominative Case-marked subject, as in (38e); and (ii) by attracting T, as in (38f). In both cases, the attracted constituent moves to the edge of c_e.[46] Continuing (38e) with the suffix [..,

[44] Examples given in this chapater are all from the English side. Fragments of Arabic, Japanese, and Persian have also received some attention. As a result, there are a few implemented actions, including some dealing with mood and scrambling, that are not listed here. (The webpage mentioned in note ~~24~~ also contains links to a small inventory of derivations of non-English examples.)

[45] We have replaced [d, *sue*] and [d, *mary*] by *Sue* and *Mary*, respectively, in (38c). This is purely for convenience of exposition; *Sue* (and *Mary*) should not be treated as heads in the derivation.

[46] In (38e), *Sue* has moved from the edge of T to the edge C. Already, at the edge of T, both of S*ue*'s unvalued features, Case and θ, have been valued. Although the stack is initially stocked with constituents with unvalued features, as discussed earlier, we do not prune away constituents that no longer possess unvalued features. One reason is that it is a chore. The second is that there is a need to keep stack items around for further non-feature-driven movement, e.g. (38e).

think, v_{unerg}, *Mary*, T, c] will derive (38b). Assuming T to C in (38f) spells out as *that*, (38f) will lead to (38a).

2.8 Computational cyclicity revisited

Long-distance agreement can be a challenge for our notion of computational cyclicity described earlier, in which Merge is closely tied to head probing. To recap, a head from the input stream gets just a single chance to probe, value features, and have features valued: precisely when it is first Merged with the current SO. Consider (39a,b):

(39) a. There seem to have been several fish caught (Chomsky 2001)
b. There seems to have been only one fish caught
c. [fish, several, catch, PRT, v∼, there, perf, v, T_{inf}, seem, v_{nop}, T, c]
d. {catch, {several!case, fish}}
e. Agree(PRT!case!ϕ, {several!case, fish})
f. Agree(there, {several!case, fish})
g. Agree(T_{inf} {several!case, fish})
h. Agree(T, {several, fish})

(39a,b) shows that long-distance ϕ-feature agreement obtains between the matrix verb and the object of the embedded clause. (See Figure 2.1 for the detailed parse computed by the model.[47]) In either case, matrix T is a probe that needs to have its unvalued uninterpretable ϕ-features valued by *several/one fish*. The initial input stream for (39a) is given in (39c), and (39d) is produced by repeated ESM using the prefix [*fish*, *several*, *catch*,..].

Next, the passive participle PRT will be merged. According to Chomsky (2001), PRT has both unvalued uninterpretable ϕ and Case features. Hence, PRT must probe, and finds *several fish*. For Agree, as in (39e), s*everal fish* can value PRT's ϕ-features, but as both of them are unvalued for Case, ~~therefore~~ PRT cannot have its Case feature valued at first Merge time. This poses a problem for our model because we would prefer not to have to search out PRT to probe again (after *several fish* receives Case from matrix T). Instead, following Fong (2014), suppose Agree can "unify" Case features. In the current model, the representation of unvalued uninterpretable Case is a term case(V), where V is an open variable. A valued Case feature is one with the open variable bound to a particular value, e.g. Nominative or Accusative. Suppose PRT and *several fish* are associated with case(V_1) and case(V_2), respectively. Then we simply unify V_1 and V_2. Essentially, they become identical but remain unvalued. When one of them is valued, the other is valued simultaneously because they have been made identical

[47] We will not explain Sobin (2014)'s v∼ here, except to mention that v∼ triggers leftwards TH/EX that raises *several fish* above the verb *catch*. Instead, we refer the reader to Ginsburg and Fong (this volume) for the details.

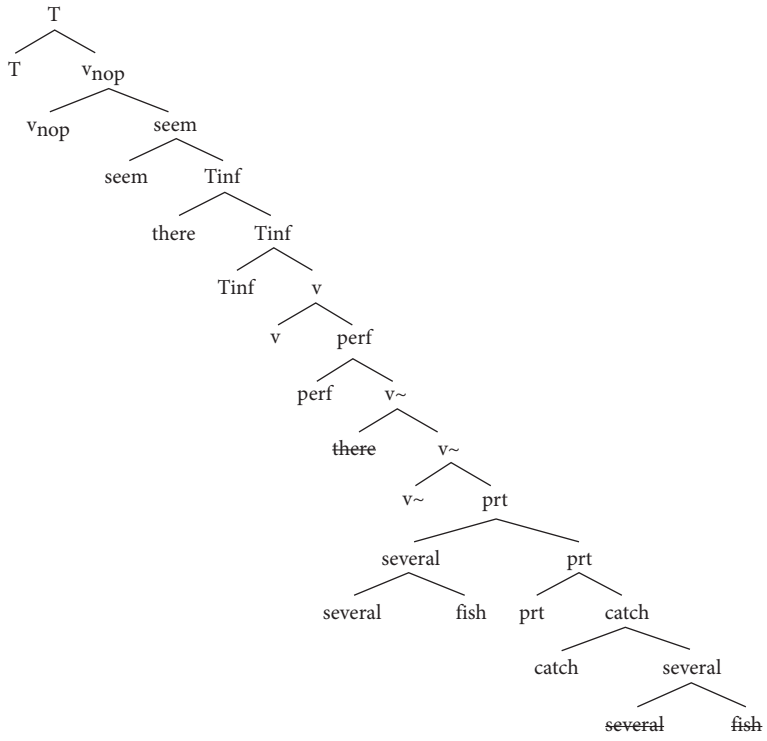

Figure 2.1 There seem to have been several fish caught

Continuing with the derivation, expletive *there* has unvalued ϕ-features.[48] If *there* is a head, it must probe, and we compute (39f), valuing the expletive's ϕ-features. Next, we need to consider Merge of non-finite T, T_{inf}. Should T_{inf} have unvalued ϕ-features, we compute (39g).[49]

Finally, Merge of finite matrix T triggers (39h). To produce the analysis in Figure 2.1, we have computed (up to) four separate Agree operations, (39e) through (39h). With unification, we have no need to search out and relaunch the most deeply embedded Agree operation. Thus, strict computational cyclicity and simplicity of the computational cycle can be maintained.

2.9 Further remarks and conclusions

We believe that building a "realistic" model of linguistic theory should be an important goal of computational modeling. Let us define "realistic" computational modeling as including automatically deriving in full detail the examples that linguists typically use. However, there is considerable scope for variation on the degree of automation. In

[48] In Chomsky (2001), expletive *there* has only unvalued person.

[49] T_{inf} seems to be able to value nullCase for PRO subjects, and so perhaps it should have ϕ-features too. V_{nop} is a dummy light verb that simply categorizes root *seem*.

our current model, we manually pre-assemble the LIs; the model is solely responsible for automatically converging on desired SOs (and rejecting those we deem illicit) in a timely manner, without going into an infinite loop, or requiring manual intervention or guidance. In a free Merge model, some of these simple goals may prove to be impractical due to the combinatorics of the problem. In the case of our model, we have a decision procedure P that is responsible for pruning the search space by selecting the right operation to perform at all possible machine states. A highly deterministic model such as the one described here is unlikely to encounter real-world limits on computational resources. The non-determinism described here is empirically productive, i.e. it is needed to generate multiple licit examples. Then it is just an empirical question whether single-action P can be maintained.

In at least two aspects, computational modeling can potentially provide useful tools that are currently unavailable. First, through detailed tracing of the operations performed, we can retrieve and compile complete step-by-step analyses (available online). Should an action or fundamental operation be modified, automation permits us to quickly and systematically determine the examples that will be affected.

Second, by bringing together theories and tweaking them to be mutually compatible, we believe we have strengthened those individual accounts. The snapshot of the MP that we have chosen to model and describe in this volume is a "mash-up" of the probe-goal framework of (Chomsky 2001), overlaid with Pesetsky and Torrego's (2001) economy-driven model of T-feature checking, a modified account of Gallego's (2006) theory of relativization—a syntactic Binding theory that extends Kayne's (2002) theory of doubling constituents—and Sobin's (2014) syntactic account of leftwards TH/EX. Space does not permit us to fully describe the modifications made, but see Ginsburg and Fong (Chapter 3 this volume) for a summary and highlights of the linguistic aspects of the implemented theory.