A Computational Guide to the Dichotomy of Features and Constraints

Thomas Graf

Stony Brook University
Department of Linguistics
mail@thomasgraf.net
http://thomasgraf.net

DGFS 2015, AG 3
March 5 2015

From the call for papers:

- Can syntactic theory avoid recourse to FFs entirely [...]?
- Can a model that eschews featural triggers be appropriately restrictive?
- Is a FF-free syntax a suitable instrument to capture optionality and obligatoriness of operations?

Answer: Yes^3! But that's not really the issue. . .

Take-Home Message

- Features and constraints are two sides of the same coin.
- We can shift the workload between them as we see fit.
- The problem is that **both are too powerful**.
- The goal is to restrict this power; pick whichever perspective is more insightful for a given problem ("anything goes").

From the call for papers:

- Can syntactic theory avoid recourse to FFs entirely [...]?
- Can a model that eschews featural triggers be appropriately restrictive?
- Is a FF-free syntax a suitable instrument to capture optionality and obligatoriness of operations?

Answer: Yes^3! But that's not really the issue...

Take-Home Message

- Features and constraints are two sides of the same coin.
- We can shift the workload between them as we see fit.
- The problem is that **both are too powerful**.
- The goal is to restrict this power; pick whichever perspective is more insightful for a given problem ("anything goes").

Outline

1. Computational Background
   - Minimalist Grammars
   - Formalizing Constraints
2. Features ≡ Constraints
   - From Features to Constraints
   - From Constraints to Features
3. Linguistic Evaluation
   - Applicability to Minimalist Syntax
   - C-Selection: The Secret Loophole

Minimalist Grammars

- This talk is about theorems and mathematically provable results. For this we need a fully explicit model of syntax.
- **Minimalist grammars** are a formalization of pre-Agree Minimalism, developed by Ed Stabler. (Stabler 1997, 2011)
- They are **completely feature-driven**.

The MG Feature Calculus

Every lexical item comes with a finite, non-empty list of features.
Feature checking must obey several non-standard properties:

- **Order** Features must be checked in the order that they appear in the list.
- **Typing** Every feature is a Merge feature or a Move feature.
- **Polarity** Every feature has either positive or negative polarity.
- **Opposition** Only identical features of opposite polarity may enter a checking relation.

Merge: Example 1

Assembling [$_{\text{DP}}$ the men]

$$\frac{\text{the}}{N^{+}\ D^{-}} \quad \frac{\text{men}}{N^{-}}$$

- Features of opposite polarities checked
- Checking triggers Merge, which builds structure on top

Merge: Example 1

Assembling [$_{DP}$ the men]

the	men
$N^+ \; D^-$	N^-

- Features of opposite polarities checked
- Checking triggers Merge, which builds structure on top

Merge: Example 1

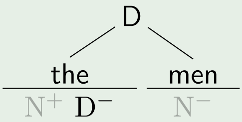
Assembling [$_{\text{DP}}$ the men]

the	men
N^+ D^-	N^-

- Features of opposite polarities checked
- Checking triggers Merge, which builds structure on top

Merge: Example 1

Assembling [$_{DP}$ the men]



- Features of opposite polarities checked
- Checking triggers Merge, which builds structure on top

Merge: Example 1

Assembling [$_{DP}$ the men]

- Features of opposite polarities checked
- Checking triggers Merge, which builds structure on top

Merge: Example 2

Assembling [$_{VP}$ the men like which men]

the	men	like	which	men
$N^+ D^-$	N^-	$D^+ D^+ V^-$	$N^+ D^-$	N^-

- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*

Merge: Example 2

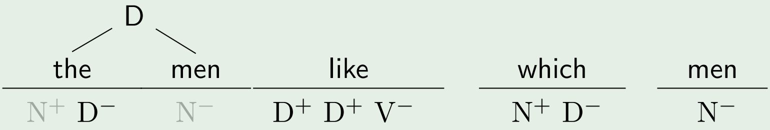
Assembling [$_{VP}$ the men like which men]

the	men	like	which	men
$N^+ \ D^-$	N^-	$D^+ \ D^+ \ V^-$	$N^+ \ D^-$	N^-

- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*

Merge: Example 2

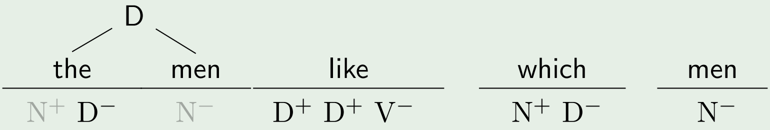
Assembling [$_{VP}$ the men like which men]



- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*

Merge: Example 2

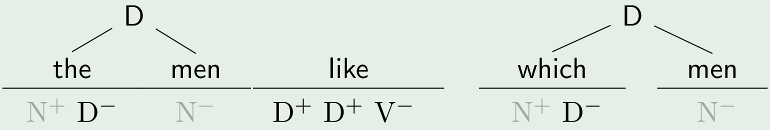
Assembling [$_{VP}$ the men like which men]



- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*

Merge: Example 2

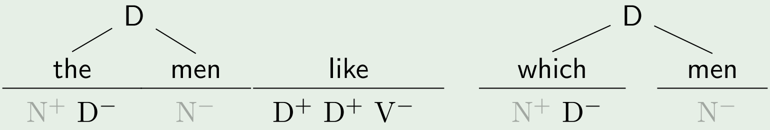
Assembling [$_{\text{VP}}$ the men like which men]



- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*

Merge: Example 2

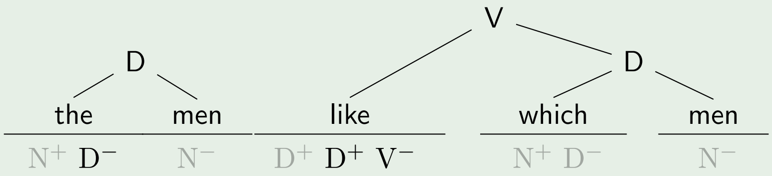
Assembling [$_{VP}$ the men like which men]



- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*

Merge: Example 2

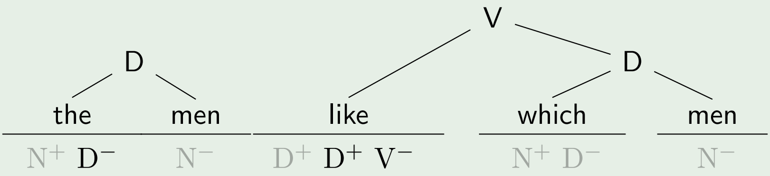
Assembling [$_{VP}$ the men like which men]



- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*

Merge: Example 2

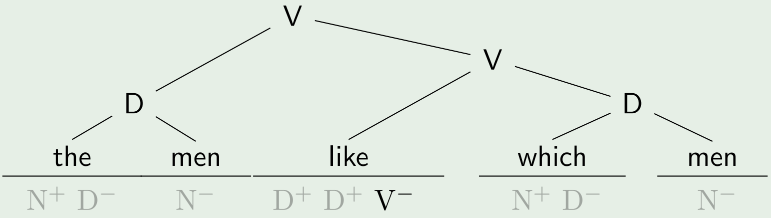
Assembling [$_{\text{VP}}$ the men like which men]



- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*

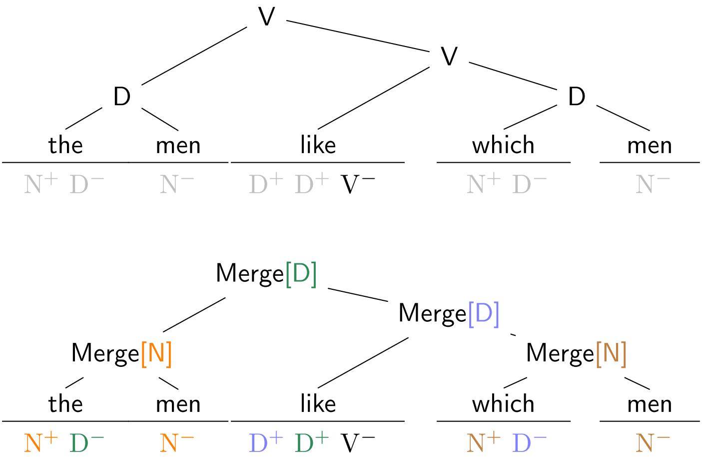
Merge: Example 2

Assembling [$_{VP}$ the men like which men]



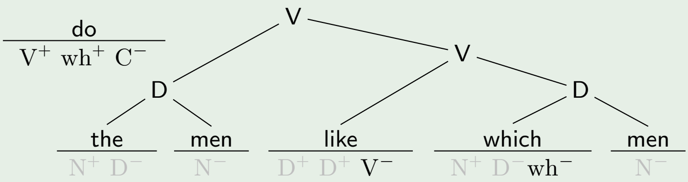
- *the* and *men* merged as before
- same steps for *which men*
- *like* merged with *which men*
- *like* merged with *the men*

Merge: Example 2 [cont.]



Move

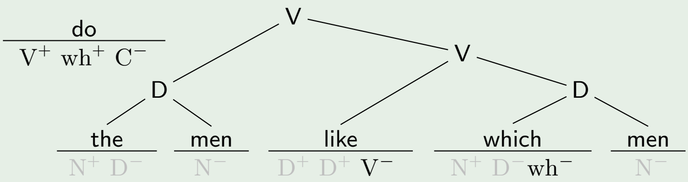
Assembling "which men do the men like?"



- Merge *do*
- Move triggered by features of opposite polarity

Move

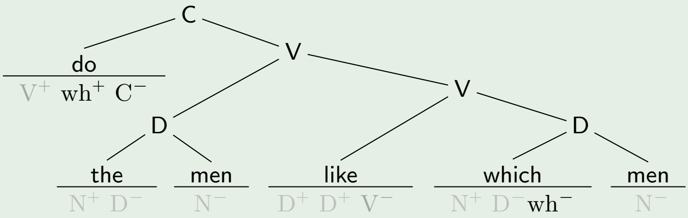
Assembling “which men do the men like?”



- Merge *do*
- Move triggered by features of opposite polarity

Move

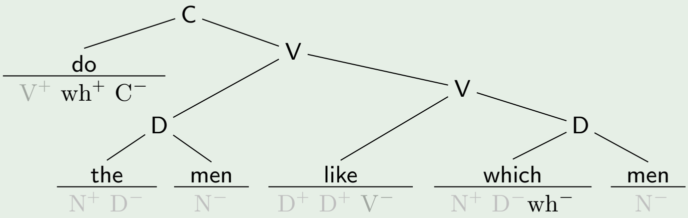
Assembling “which men do the men like?”



- Merge *do*
- Move triggered by features of opposite polarity

Move

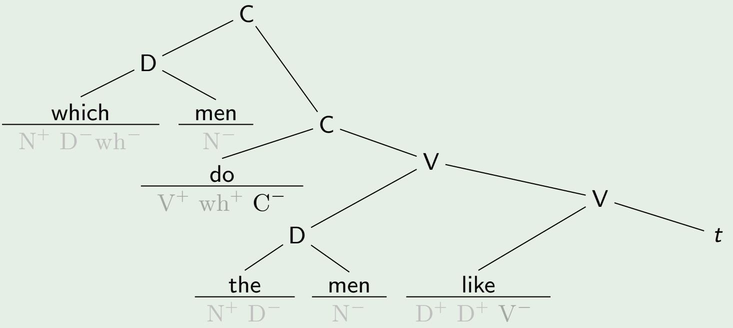
Assembling "which men do the men like?"



- Merge *do*
- Move triggered by features of opposite polarity

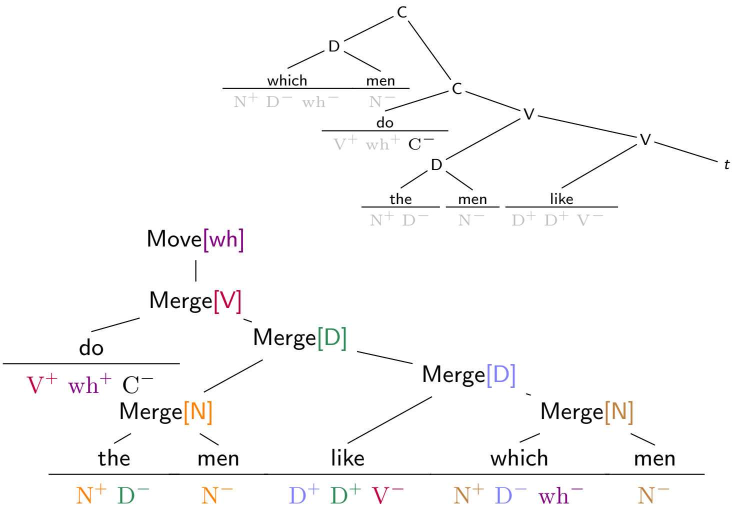
Move

Assembling "which men do the men like?"



- Merge *do*
- Move triggered by features of opposite polarity

Derivation Trees with Move



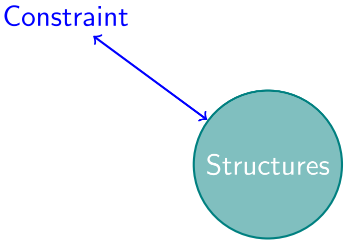
Formalizing Constraints

- Every (non-violable) constraint can be identified with the set of structures that satisfy the constraint.
- Every set of structures can be identified with a logical formula that holds of these and only these structures.
- Hence **constraints are logical formulas**.
  (Kracht 1995; Rogers 1998; Potts 2001; Pullum 2007; Graf 2011, 2013)

Constraint

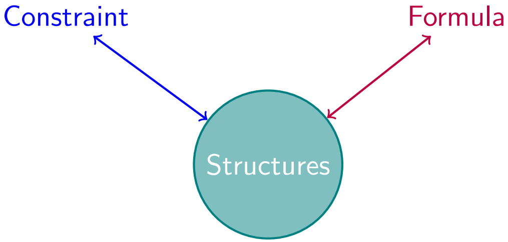
Formalizing Constraints

- Every (non-violable) constraint can be identified with the set of structures that satisfy the constraint.
- Every set of structures can be identified with a logical formula that holds of these and only these structures.
- Hence **constraints are logical formulas**.
  (Kracht 1995; Rogers 1998; Potts 2001; Pullum 2007; Graf 2011, 2013)



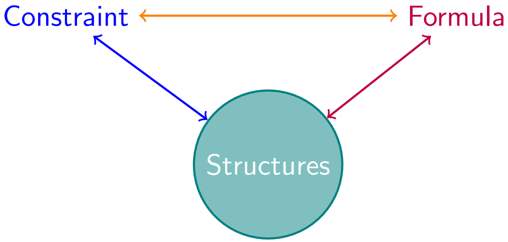
Formalizing Constraints

- Every (non-violable) constraint can be identified with the set of structures that satisfy the constraint.
- Every set of structures can be identified with a logical formula that holds of these and only these structures.
- Hence **constraints are logical formulas**.
  (Kracht 1995; Rogers 1998; Potts 2001; Pullum 2007; Graf 2011, 2013)



Formalizing Constraints

- Every (non-violable) constraint can be identified with the set of structures that satisfy the constraint.
- Every set of structures can be identified with a logical formula that holds of these and only these structures.
- Hence **constraints are logical formulas**.
  (Kracht 1995; Rogers 1998; Potts 2001; Pullum 2007; Graf 2011, 2013)



Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z[\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z [\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z [\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

"For every x

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z [\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

“For every x that is an anaphor

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \land \text{DP}(y) \land \exists Z[\text{bind-dom}(Z, x) \land y \in Z] \big] \Big]$$

“For every x that is an anaphor it holds that

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z[\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

"For every x that is an anaphor it holds that

there is a y

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z [\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

"For every x that is an anaphor it holds that
there is a y that c-commands x

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z [\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

"For every x that is an anaphor it holds that
there is a y that c-commands x and

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z [\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

"For every x that is an anaphor it holds that
there is a y that c-commands x and is labeled DP,

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z[\text{bind-dom}(Z, x) \wedge y \in Z]\big]\Big]$$

"For every x that is an anaphor it holds that
there is a y that c-commands x and is labeled DP, and

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP
within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z [\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

"For every x that is an anaphor it holds that
there is a y that c-commands x and is labeled DP, and
there is a set Z of nodes such that

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z[\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

"For every x that is an anaphor it holds that
there is a y that c-commands x and is labeled DP, and
there is a set Z of nodes such that
Z is the binding domain of x

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z[\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

"For every x that is an anaphor it holds that
there is a y that c-commands x and is labeled DP, and
there is a set Z of nodes such that
Z is the binding domain of x and

Example: A First-Order Formula for Principle A

Principle A (slightly simplified)

Every anaphor must be c-commanded by some DP within its binding domain.

$$\forall x \Big[\text{anaphor}(x) \rightarrow \exists y \big[\text{c-com}(y, x) \wedge \text{DP}(y) \wedge \exists Z [\text{bind-dom}(Z, x) \wedge y \in Z] \big] \Big]$$

"For every x that is an anaphor it holds that
there is a y that c-commands x and is labeled DP, and
there is a set Z of nodes such that
Z is the binding domain of x and Z contains y."

Logics for Constraints

- The logic in the previous example is first-order logic with set quantification, aka **monadic second-order logic (MSO)**.
- MSO allows us to talk about
  - node labels (including feature structures),
  - local and non-local dependencies between nodes,
  - domains within which dependencies must hold.
- This makes MSO sufficiently powerful for all syntactic constraints, including even transderivational ones. (Graf 2012, 2013)
- In the literature but beyond MSO: identity of meaning
- Henceforth “constraint” = MSO-definable constraint

Interim Summary

- **MGs**
  - MGs are a purely feature-driven formalism.
  - Every MG can be identified with its set of well-formed derivations.
- **Constraints**
  - Every constraint can be identified with its set of licensed structures.
  - Consequently, constraints can be equated with logical formulas.
  - MSO formulas are powerful enough for syntax.

A First Connection

Every MG can be identified with an MSO constraint picking out its set of well-formed derivations.
$\Rightarrow$ representational view of MGs, but not feature-free

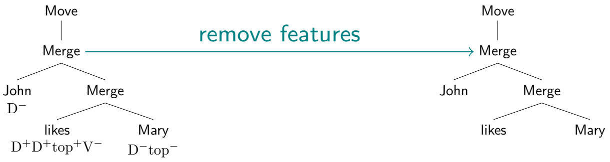
Outline

Features are Inessential

Feature Removal Preserves Output Language

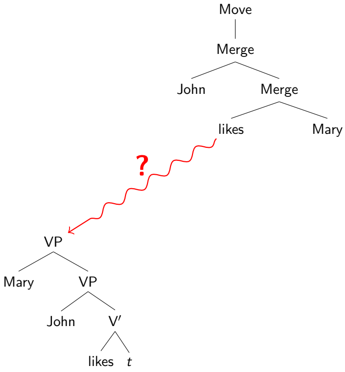
Every MG can be made feature-free without altering the set of generated phrase structure trees.

- Let D be the set of derivation trees for some MG G.
- Let **remove features** be the mapping that removes all feature annotations from every derivation.
- Applying **remove features** to D yields a set D' of trees that is definable in MSO $\Rightarrow$ D' defines an MSO constraint



Spell-Out Without Features

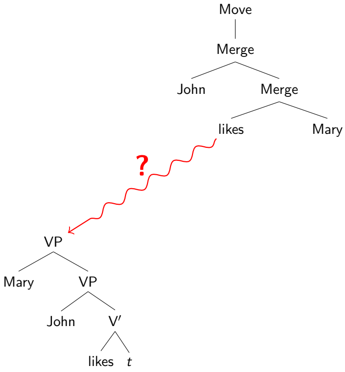
But how do we get the intended mapping from derivations to phrase structure trees if there are no features?



Spell-Out Without Features

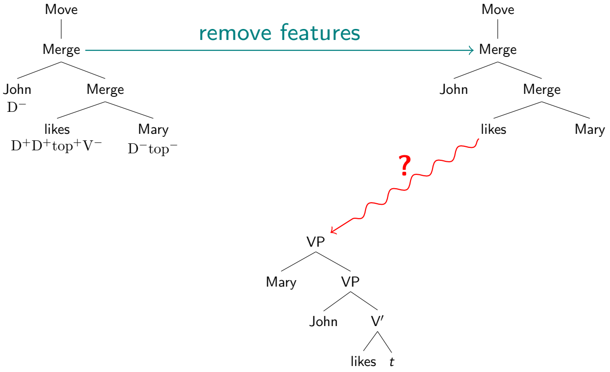
But how do we get the intended mapping from derivations to phrase structure trees if there are no features?

Answer: construct feature-free spell-out from feature-based one



Spell-Out Without Features

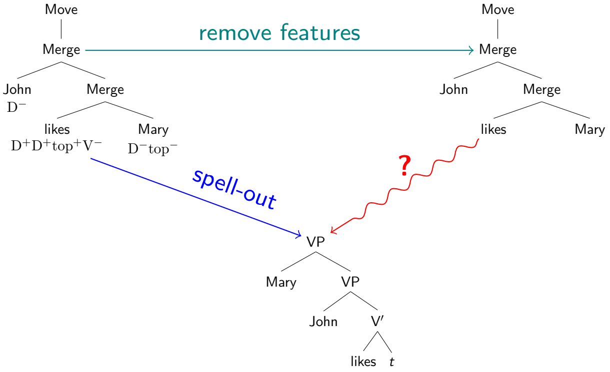
But how do we get the intended mapping from derivations to phrase structure trees if there are no features?
Answer: construct feature-free spell-out from feature-based one



Spell-Out Without Features

But how do we get the intended mapping from derivations to phrase structure trees if there are no features?

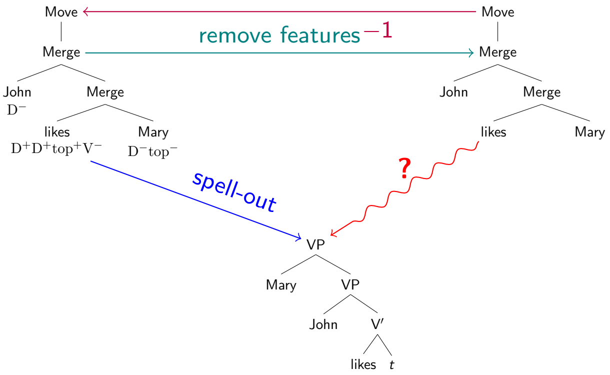
Answer: construct feature-free spell-out from feature-based one



Spell-Out Without Features

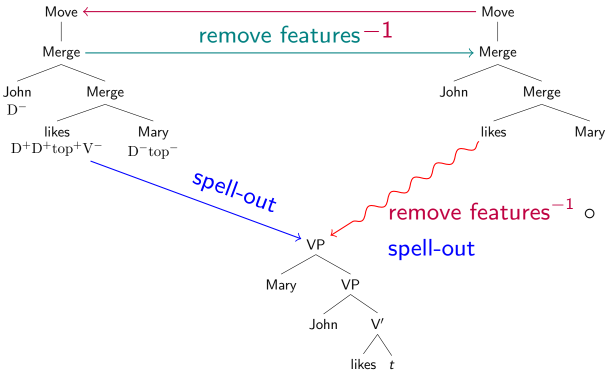
But how do we get the intended mapping from derivations to phrase structure trees if there are no features?

Answer: construct feature-free spell-out from feature-based one



Spell-Out Without Features

But how do we get the intended mapping from derivations to phrase structure trees if there are no features?
Answer: construct feature-free spell-out from feature-based one



Feature-free Spell-Out is Feature-Free

Feature-free spell-out does not construct any intermediate, feature-annotated derivations. It is a direct mapping from feature-free derivations to phrase structure trees.

A Non-Linguistic Analogy

Let $add(x) = x + 1$ and $sub(x) = x - 1$. Then we have

x	$add(x)$	$sub(add(x))$
1	2	1
2	3	2
3	4	3
$\vdots$		

Note that $sub(add(x)) = x$ for every x. So the composite function $sub \circ add$ is just the identity function, it never computes the intermediate value $add(x)$.

Summary: Why Features do not Matter

- The MG feature calculus does two things:
  1. define a set of well-formed derivation trees,
  2. control the translation from derivation trees to phrase structure trees.
- MSO constraints can determine well-formedness without the explicit information provided by features.
- Similarly, spell-out can be replaced by a suitably constrained translation that does not need features.
- **Generalization**
  Features are a way of lexicalizing information, but we can also **delexicalize** this information back into constraints.

Constraints can be Lexicalized

Grammar Precompilation Preserves Output Language

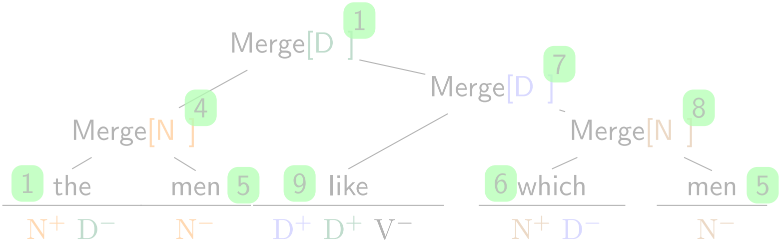
Every MSO constraint can be precompiled into the grammar without altering the set of generated phrase structure trees.

Intuition

- Decompose the constraint into a sequence of local constraints.
- Represent the information flow between the local constraints as special node labels in the derivation tree.
- Lexicalize the information flow by pushing the new labels into the category features.
- C-selection via Merge now enforces all local constraints, and by extension also the original constraint.

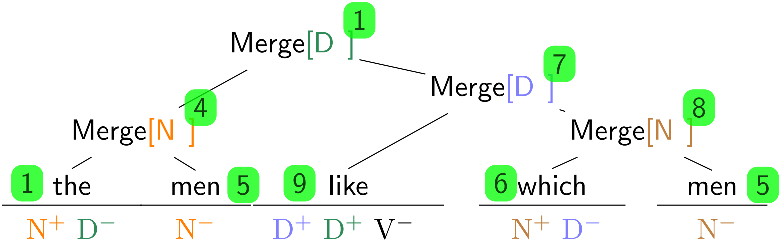
An Example Sketch

- **Decomposition:** translate MSO constraint into equivalent finite-state tree automaton
- **Representation:** induce state assignment of automaton



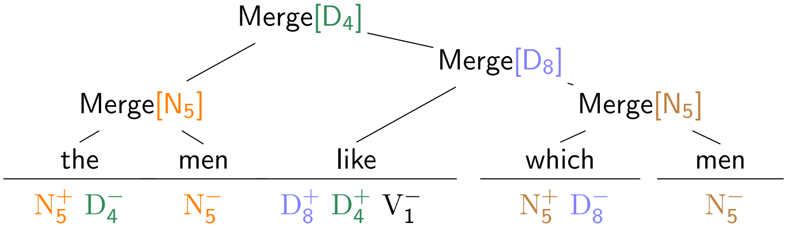
An Example Sketch

- **Decomposition:** translate MSO constraint into equivalent finite-state tree automaton
- **Representation:** induce state assignment of automaton



An Example Sketch

- **Decomposition:** translate MSO constraint into equivalent finite-state tree automaton
- **Representation:** induce state assignment of automaton



Summary: Why Features Can Replace Constraints

- MSO constraints are the most powerful class of constraints whose behavior can still be understood as the interaction of simple local dependencies.
- These local dependencies fall within the locality domain of c-selection/subcategorization.
- Hence they can be **lexicalized** via Merge features.

Equivalence of Features and Constraints

Let C be a dependency over Minimalist derivations. Then C is an MSO constraint iff it can be enforced via the MG feature calculus.

Summary: Why Features Can Replace Constraints

- MSO constraints are the most powerful class of constraints whose behavior can still be understood as the interaction of simple local dependencies.
- These local dependencies fall within the locality domain of c-selection/subcategorization.
- Hence they can be **lexicalized** via Merge features.

Equivalence of Features and Constraints

Let C be a dependency over Minimalist derivations. Then C is an MSO constraint iff it can be enforced via the MG feature calculus.

Outline

Do These Findings Also Hold for Minimalism?

Complaint 1: MG Deviations from Minimalist Syntax

- **Operations:** no Agree, only phrasal movement
- **Feature calculus:** order, typing, polarity, opposition

All these differences are **irrelevant**. The equivalence between features and MSO constraints holds for every formalism that satisfies the following properties:

- There is some lexicalized mechanism for subcategorization.
- The mechanism distinguishes complements from specifiers.

Both properties are indispensable for even the most basic facts:

(1) a. $[[_{vP}\ [_{DP}\ \text{John}]\ [_{v'}\ v\ [_{VP}\ \text{slept}]]]$

b. * $[[_{vP}\ [_{VP}\ \text{slept}]\ [_{v'}\ v\ [_{DP}\ \text{John}]]]$

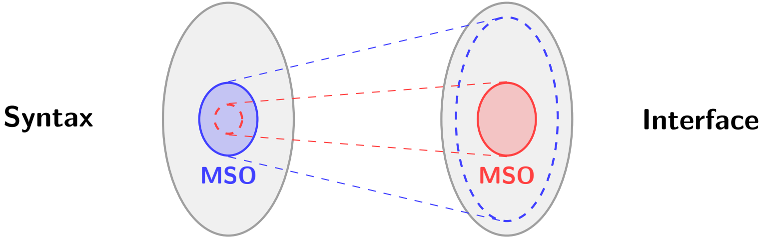
Interface Constraints

Complaint 2: Locus of Constraints

Constraints in Minimalism apply at the interfaces,
not during the derivation.

This actually **increases the power of features**.

- Every MSO interface constraint can be translated into an MSO constraint over derivations, but not the other way round.
- Hence the feature calculus can encode interface constraints that are not even MSO-definable.



Restrictions on the Feature System

Complaint 3: Category Refinement

The equivalence fails if the set of category features is fixed.

- Actually the set can be fixed as long as it is big enough for the constraints of interest. Every wide-coverage grammar nowadays has hundreds of parts of speech.
- More generally, this simply **begs the question**. Syntacticians presuppose a fixed set of categories and let the constraints vary across languages, but the equivalence result shows that this is neither an empirical nor a conceptual necessity.

C-Selection: The Secret Loophole

Simple Corollary of Feature-Constraint Equivalence

A formalism with c-selection can express every MSO constraint.

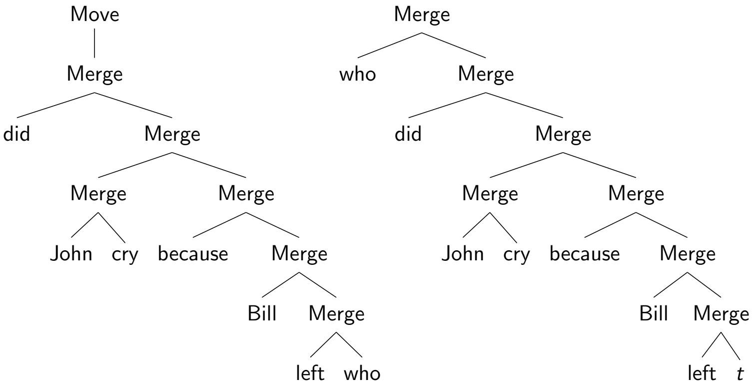
Problem 1: MSO is too powerful!
Here's a list of unnatural MSO constraints:

- An anaphor must c-command its antecedent.
- The number of nodes must be a multiple of 17.
- A derivation must obey Principle A or B, but not both.

Problem 2: MSO constraints can bleed other constraints!

Move as MSO-Controlled Merge

Island constraints can be circumvented via Merge.
(cf. resumptive pronoun analyses)



What the Feature-Constraint Equivalence is Really About

- Dependencies can be encoded locally via features or non-locally via constraints.
- We can switch between these perspectives as we see fit.
- There may ultimately be reasons to prefer one over the other in all cases, but this is a premature question.
  (Personally, I don't think there is a best encoding.)
- Right now, the most pressing issue is limiting the class of definable dependencies, no matter how.

Examples:

constraints	Contiguity theory (Richards 2014)
features	Syntactic buffers (Müller 2014)
hybrid	Feature algebras for morpho-syntax (Graf 2014)

References I

Graf, Thomas. 2011. Closure properties of minimalist derivation tree languages. In *LACL 2011*, ed. Sylvain Pogodalla and Jean-Philippe Prost, volume 6736 of *Lecture Notes in Artificial Intelligence*, 96–111. Heidelberg: Springer.

Graf, Thomas. 2012. Reference-set constraints as linear tree transductions via controlled optimality systems. In *Formal Grammar 2010/2011*, ed. Philippe de Groote and Mark-Jan Nederhof, volume 7395 of *Lecture Notes in Computer Science*, 97–113. Heidelberg: Springer.

Graf, Thomas. 2013. *Local and transderivational constraints in syntax and semantics*. Doctoral Dissertation, UCLA.

Graf, Thomas. 2014. Feature geometry and the person case constraint: An algebraic link. In *Proceedings of CLS 50*. To appear.

Kracht, Marcus. 1995. Is there a genuine modal perspective on feature structures? *Linguistics and Philosophy* 18:401–458.

Müller, Gereon. 2014. *Syntactic buffers*. Linguistische Arbeitsberichte.

Potts, Christopher. 2001. Three kinds of transderivational constraints. In *Syntax at Santa Cruz*, ed. Séamas Mac Bhloscaidh, volume 3, 21–40. Santa Cruz: Linguistics Department, UC Santa Cruz.

Pullum, Geoffrey K. 2007. The evolution of model-theoretic frameworks in linguistics. In *Model-Theoretic Syntax @ 10*, ed. James Rogers and Stephan Kepser, 1–10.

References II

Richards, Norvin. 2014. Contiguity theory. Unpublished Ms., MIT.

Rogers, James. 1998. *A descriptive approach to language-theoretic complexity*. Stanford: CSLI.

Stabler, Edward P. 1997. Derivational minimalism. In *Logical aspects of computational linguistics*, ed. Christian Retoré, volume 1328 of *Lecture Notes in Computer Science*, 68–95. Berlin: Springer.

Stabler, Edward P. 2011. Computational perspectives on minimalism. In *Oxford handbook of linguistic minimalism*, ed. Cedric Boeckx, 617–643. Oxford: Oxford University Press.