A quantifier-based approach to NPI-licensing typology: Empirical and computational investigations

Dissertation defense

Mai Ha Vu

October 4, 2019

Outline

Negative Polarity Items (NPIs)

Definition (Negative Polarity Item)

A negative polarity item α is an expression whose distribution is limited by sensitivity to some semantic property β. β must at least include negation.

It shows the following contrast:

(1) a. Nancy does not want anything.
b. * Nancy wants anything.

The questions addressed in the thesis

This thesis examines the *Quantifier-based approach* to Negative Polarity Item licensing typology, from two perspectives:

- **Empirical**: How adequate is this theory in explaining cross-linguistic differences in NPI-behavior?
- **Computational**: How computationally complex are the constraints in this approach?

Today, I focus on my computational results.

Goals of this presentation

- Introduce the quantifier-based approach to NPI-licensing typology
- Provide the necessary formal background
- Demonstrate how a specific theory can be modeled with computational tools
- Discuss the effects of choosing a theory on the computational results

Outline

Quantifier-based approach

- In this thesis, I adopt a quantifier-based approach to NPI-licensing following the ideas in Giannakidou (2000).
- **Observation**: a sentence like '*I did not see anybody*' could be expressed semantically in one of two ways:

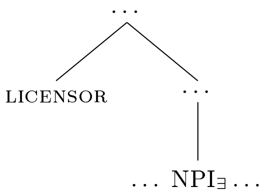
(2) $\forall x[\text{person}(x) \rightarrow \neg\,\text{see}(I, x)]$

(3) $\neg\exists x[\text{person}(x) \wedge \text{see}(I, x)]$

- **Proposal**: NPIs can be expressed with different quantifiers, and that predicts their syntactic behavior
  - If they are universal quantifiers (∀-NPI), they have to take scope above negation at Logical Form (LF)
  - If they are existential quantifiers (∃-NPI), they have to take scope below negation (or NPI-licensor) at LF

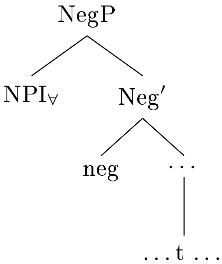
Licensing ∃-NPIs

- Existentially quantified NPIs must be in the scope of their licensor at LF
- Scope-domain is calculated through c-command
- A node c-commands its sibling and all the nodes its sibling dominates → A node takes scope over everything it c-commands



Licensing ∀-NPIs

- ∀-NPIs must scope over negation at LF
- They do so by undergoing Quantifier Raising (QR) and attach to NegP



Cross-linguistic differences in syntactic behavior

	∃-NPIs	∀-NPIs
Can appear higher than licensor	no	yes
Long-distance licensing	yes	no
Fragment answers	no	yes
Sensitivity to islands	no	yes
Examples	English *any*-NPIs	Hungarian *se*-NPIs

∀-NPIs must scope over negation → They can be higher on the surface then their licensor and not reconstruct:

(4) a. * Anybody did not see the movie.

b. Sen-ki nem látta a film-et.
NPI-who NEG see-PST.3SG the movie-ACC
'Anybody did not see the movie. = Nobody saw the movie.'

Cross-linguistic differences in syntactic behavior

	∃-NPIs	∀-NPIs
Can appear higher than licensor	no	yes
Long-distance licensing	yes	no
Fragment answers	no	yes
Sensitivity to islands	no	yes
Examples	English *any*-NPIs	Hungarian *se*-NPIs

QR is clause-bounded → ∀-NPIs must be licensed by clause-mate negation:

(11) Some man said every woman visited him. $\exists \gg \forall$, $*\forall \gg \exists$

(12) a. Sue doesn't think that Joe would meet with anyone.

b. * Sue nem gondol-ta, hogy Joe találkoz-na sen-ki-vel.
Sue NEG think-PST.3SG that Joe meet-COND.3SG NPI-who-COM
'Sue doesn't think that Joe would meet with anyone.'

Cross-linguistic differences in syntactic behavior

	∃-NPIs	∀-NPIs
Can appear higher than licensor	no	yes
Long-distance licensing	yes	no
Fragment answers	no	yes
Sensitivity to islands	no	yes
Examples	English *any*-NPIs	Hungarian *se*-NPIs

∀-NPIs can raise above negation, and have the rest of the sentence elided for fragment answers:

(19) a. Who did you see? *Anyone ~~I did not see~~ .

b. Ki-t lát-tál? Sen-ki-t ~~nem~~ ~~lát-t-am~~ .
who-ACC see-PST.2SG NPI-who-ACC NEG see-PST-1SG
'Who did you see? Nobody.'

Cross-linguistic differences in syntactic behavior

	∃-NPIs	∀-NPIs
Can appear higher than licensor	no	yes
Long-distance licensing	yes	no
Fragment answers	no	yes
Sensitivity to islands	no	yes
Examples	English *any*-NPIs	Hungarian *se*-NPIs

Island sensitivity indicates movement (including QR) → ∀-NPIs are sensitive to islands, because they undergo QR:

(26) A student ate a slice of pizza <and/or every slice of cake>. $\exists \gg \forall, *\forall \gg \exists$

(27) a. Sam didn't eat <beans or anything>.

b. Most people eat beans and rice and beans and toast, but he doesn't eat <beans and anything>! (p.c. Bruening)

c. * Jancsi nem eszik <bab-ot és/vagy sem-mi-t>.
Jancsi NEG eat bean-ACC and/or NPI-what-ACC
'Jancsi doesn't eat beans and/or anything.'

Locality of QR revisited

- Newer experimental evidence suggests that QR is not *actually* clause-bound, but might be a processing effect (Wurmbrand, 2018)
- Hungarian shows constrast between covert and overt movement:

(28) * Sue nem gondol-ta, hogy Joe találkoz-na sen-ki-vel.
Sue NEG think-PST.3SG that Joe meet-COND.3SG NPI-who-COM
'Sue doesn't think that Joe would meet with anyone.'

(29) Sue sen-ki-vel$_i$ nem gondol-ta, hogy Joe találkoz-na t$_i$.
Sue NPI-who-COM NEG think-PST.3SG that Joe meet-COND.3SG
'Sue doesn't think that Joe would meet with anyone.'

(30) Anna sen-ki-vel$_i$ nem hall-otta, hogy Sue meg ígér-te,
Anna NPI-who-COM NEG hear-PST.3SG that Sue PRT promise-PST.3SG
hogy találkoz-na t$_i$.
that meet-COND.3SG
'Anna didn't hear that Sue promised that she would meet with anyone.'

Overview of the constraints

- ∃-NPIs: must be c-commanded by negation
- ∀-NPIs: must c-command negation through movement, AND covert-movement is clause-bound

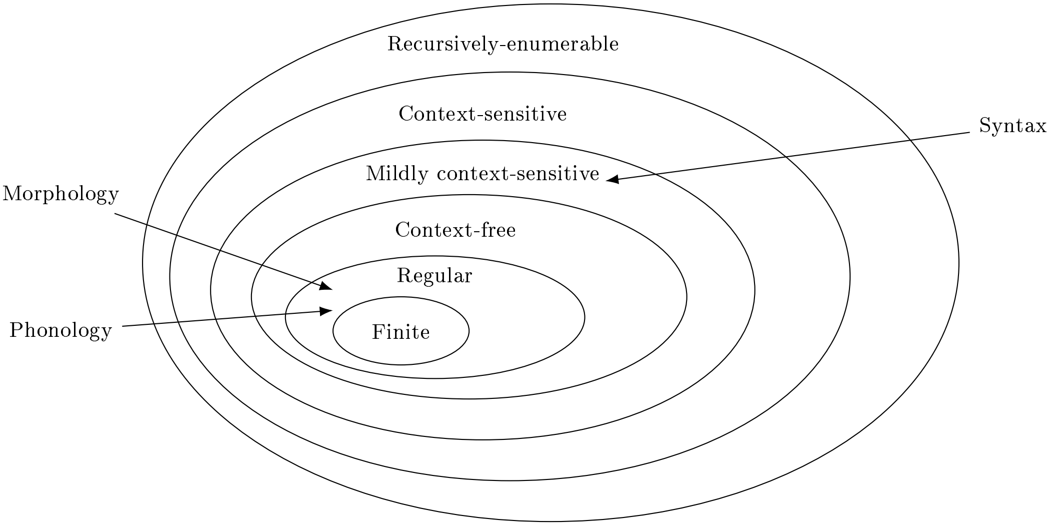
Outline

Computational complexity

- A *computer* uses an *algorithm* to generate an *output*
- If the human cognitive faculty is a type of computer, then it uses *grammar* to generate *strings in natural language*
- Computational complexity measures the complexity of the grammar: how mathematically powerful are the tools needed to describe it?
- The actual grammar of natural language is unobservable directly → we have to rely on the output to infer the grammar, and the output is a string

→ **Overarching question**: Based on the string outputs, how complex are the most complex patterns in different modules of natural language?

How complex is natural language?



Syntax is mildly context-sensitive?

Joshi's (1985) conjecture, based on Shieber's (1985) observation:

- Swiss German cross-serial dependency is a mildly context-sensitive pattern ($a^n b^m c^n d^m$)

(31) ...mer em Hans es huus hälfed aastriche (Shieber, 1985)
...we Hans-DAT the house helped paint
'We helped Hans paint the house.'

- Syntax must be powerful enough to generate such patterns
- BUT: this assumes that the relevant data structure output by the syntax is a string – the *string* output of syntax is mildly context sensitive
- What if the we take the relevant data structure to be *trees*?

The complexity of syntactic trees

- Thatcher (1967): Regular tree languages yield Context-Free string-languages
  - This brings down the complexity of most syntactic constraints to the regular class of languages
  - But, it still does not cover Mildly Context-Sensitive patterns
- Morawietz's (2003) Two-step approach: describe syntax in two parts
  - Constraints that restrict the syntactic derivation
  - Functions to map the derivation to the output(s)

  → if both are Regular, then they can generate Mildly Context-Sensitive string languages

For the thesis, I focus on the first component, on restricting the syntactic derivation, and encode it using *derivation trees* in the Minimalist Grammars (MGs) framework.

Minimalist Grammars

- An explicit formalization of Minimalist syntax, first described (Stabler, 1997)
- Two components: lexicon and operations
- **Lexicon**: a finite set of Lexical Items (LIs), that consist of a phonological component, a semantic component, and strictly ordered features
  - Example: [which :: =n d −wh]
- **Operations**: originally `Merge` and `Move`. In this thesis, I add
  - `S(emantic)-move` for movement only at LF
  - `P(honological)-move` for movement only at PF
  - `Cluster` for movement of multiple items of the same type (e.g. multiple wh-movement), after Sabel (2001); Grewendorf (2001), formalized for MGs in Gärtner and Michaelis (2010)

Features

Features have four attributes:

- Name: what is the feature called?
- Operation: what operation is the feature associated with?
- Polarity: does the feature have negative polarity or positive polarity?
- Representation: Does it go with an operation that takes place at PF, LF, or both?

shorthand	ν	ω	π	ρ
f	f	`Merge`	−	[+sem,+phon]
=f	f	`Merge`	+	[+sem,+phon]
−f	f	`Move`	−	[+sem,+phon]
+f	f	`Move`	+	[+sem,+phon]
$-_s$f	f	`Move`	−	[+sem,−phon]
$+_s$f	f	`Move`	+	[+sem,−phon]
$-_p$f	f	`Move`	−	[−sem,+phon]
$+_p$f	f	`Move`	+	[−sem,+phon]

Example: [which :: =n d −wh]

- d means that *which* has the category feature d
- =n means that *which* selects for an LI whose category feature is n
- −wh means that *which* has a wh movement licensee feature on it that will have to be satisfied by `Move`

Derivation trees vs. Derived phrase structure trees

- Derivation trees show the *process* of the derivation, rather than the output of it
- Instead of category labels, trees are labeled with the operation (which can be inferred from the features of the LIs)

Let's build a tree

(32) Mary likes the car.

Figure 1: Phrase-structure tree

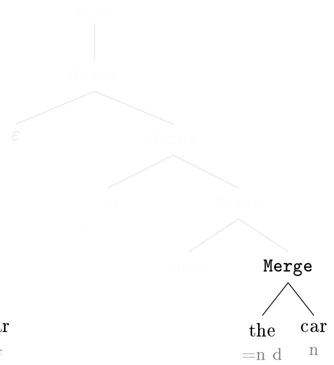


Figure 2: Derivation tree

Let's build a tree

(33) Mary likes the car.

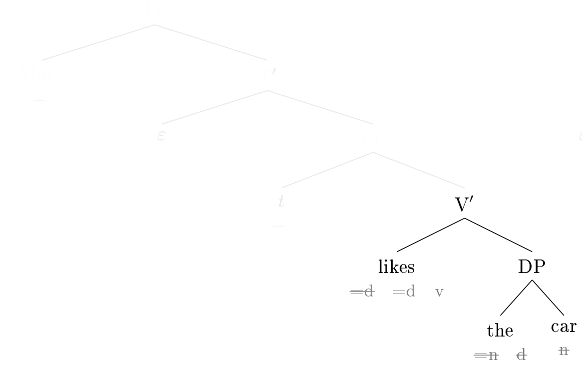


Figure 1: Phrase-structure tree

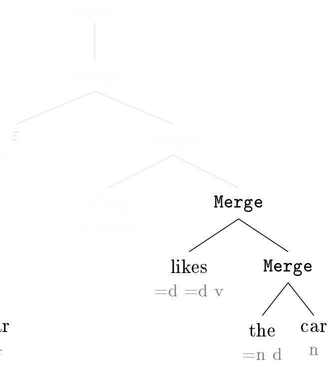


Figure 2: Derivation tree

Let's build a tree

(34) Mary likes the car.

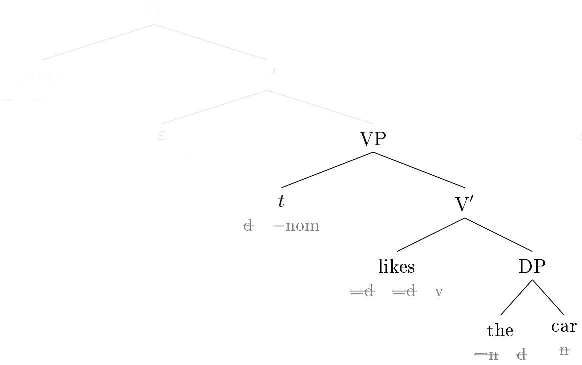


Figure 1: Phrase-structure tree

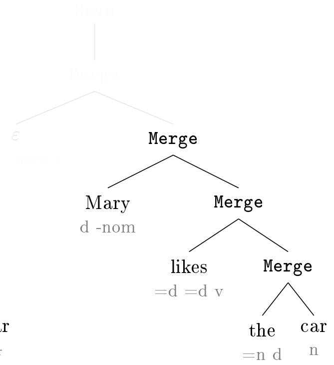


Figure 2: Derivation tree

Let's build a tree

(35) Mary likes the car.

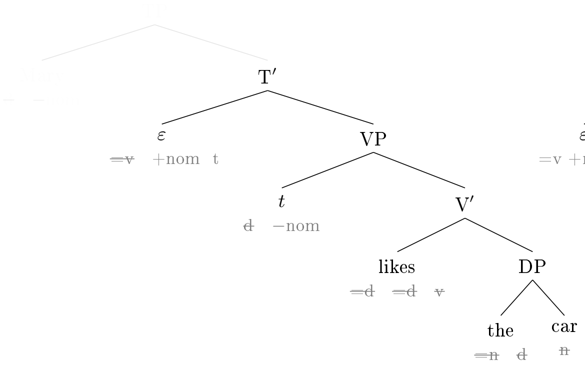


Figure 1: Phrase-structure tree

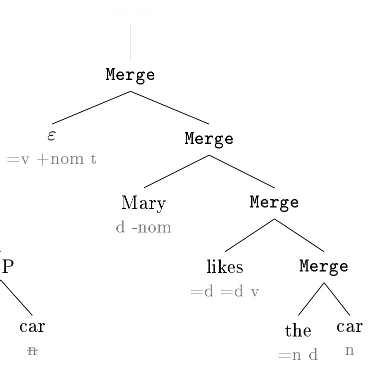


Figure 2: Derivation tree

Let's build a tree

(36) Mary likes the car.

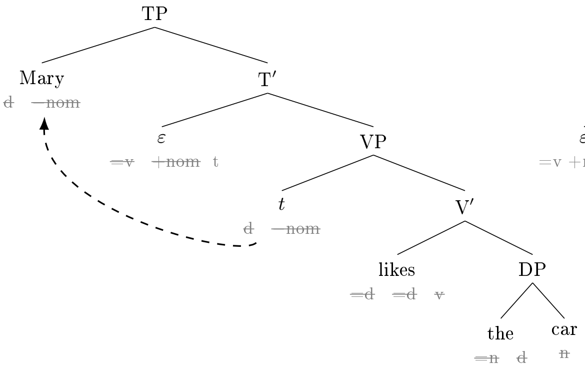


Figure 1: Phrase-structure tree

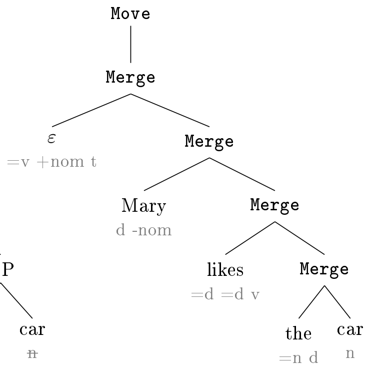
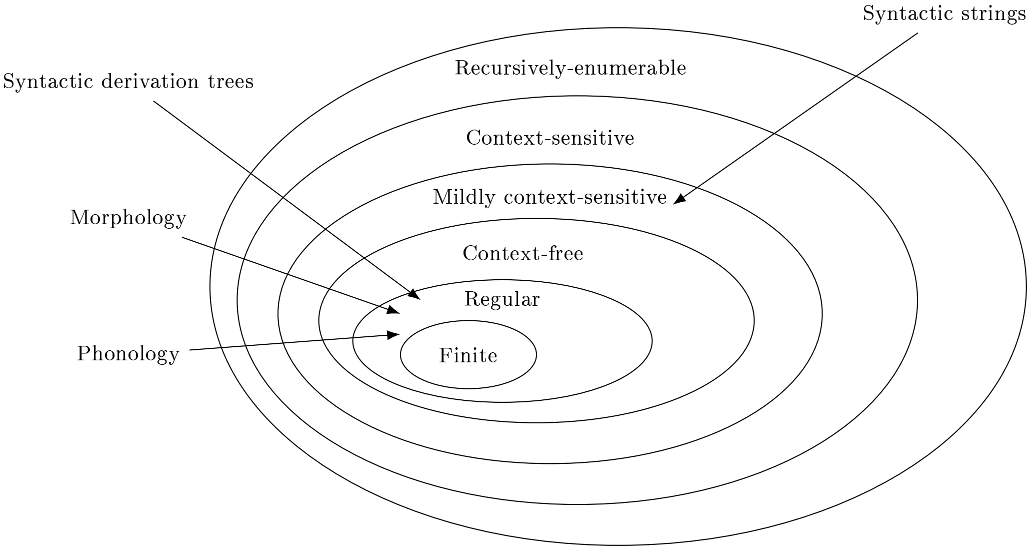


Figure 2: Derivation tree

Back to complexity

- Well-formed MGs derivation trees are regular (Kobele et al., 2007)
- Ties in to the question of *representation* vs. *logical constraints* (Jardine, 2016)
  - If the output of syntax is represented as a string-language, then we need high complexity in the logical constraints
  - If the output of syntax is represented as a tree-language, we can significantly lower the complexity of the logical tools needed to describe the patterns

The complexity of natural language - revised



Cognitive parallelism hpyothesis

- Recent work in phonology has found that most phonological patterns are not only regular, they are *subregular* (Chandlee, 2014; Jardine, 2016)
- Basic syntactic operations, such as `Merge` and `Move` can also be described with *subregular* constraints (Graf and Heinz, 2015)

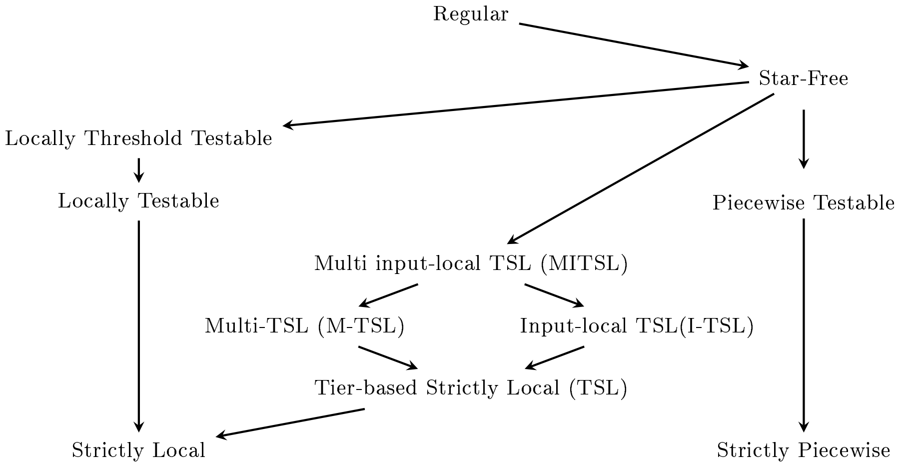
→ Proposal by (Graf et al., 2018):

Definition (Cognitive parallelism hypothesis)

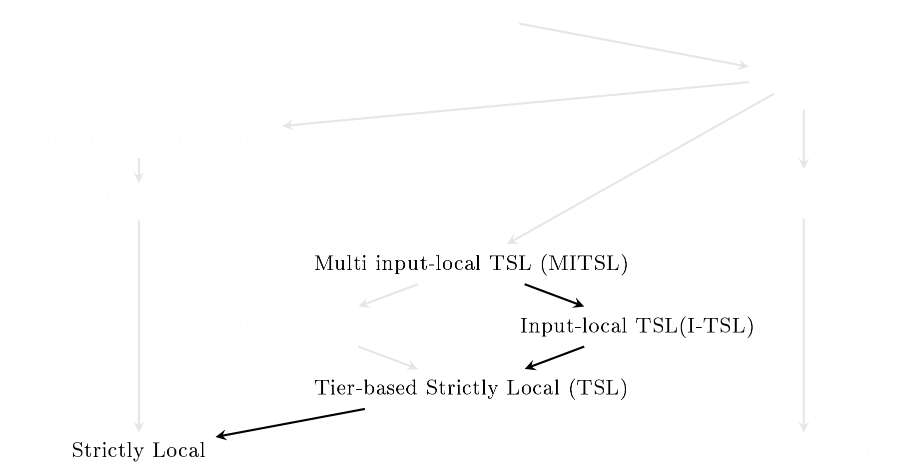
Phonology, morphology, and syntax have the same subregular complexity over their respective structural representations.

→Can other dependencies in syntax, such as NPI-licensing, also be *subregular*?

The subregular hierarchy



The subregular hierarchy



Strictly Local languages

Intuitive description: List possible substructures of k size (or equivalently, list banned substructures of k size)

Example (SL grammar over strings)

(from Graf et al. (2018))

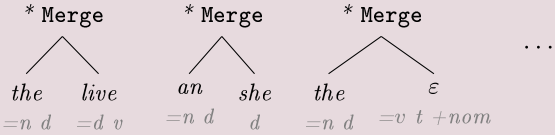
- *German word final devoicing: forbid voiced segments in the end of the string*
- ***SL Grammar:*** *d$, *z$, *v$, etc.
- The grammar correctly rules out *$ra**d$** and accepts $ra**t$**

Strictly Local languages

Intuitive description: List possible substructures of k size (or equivalently, list banned substructures of k size)

Example (SL grammar over trees)

- *Merge for nouns: one of the* Merge *node's LI child must have an =n selector feature, and its other LI child must have an n category feature*
- *The grammar lists banned subtrees of bound depth (in this case, 2)*



Tier-based languages over strings

Intuitive description:

- Project a tier
- Apply constraints over the tier

Tier-based languages over strings

Intuitive description:

- Project a tier
- Apply constraints over the tier

Tier-based Strictly Local languages

Tier-based languages over strings

Intuitive description:

- Project a tier
- Apply constraints over the tier

Tier-based Strictly Local languages

- Project a tier with the help of an erasing function – erase all nodes that are irrelevant for the constraint

Example (TSL over strings)

- *String: bibobua*
- *Tier: T={a,e,i,o,u}*
- *Erasing function yields: ioua*

Tier-based languages over strings

Intuitive description:

- Project a tier
- Apply constraints over the tier

Tier-based Strictly Local languages

- Project a tier with the help of an erasing function – erase all nodes that are irrelevant for the constraint
- Apply SL constraints over the tier

Example (TSL over strings)

- *String: bibobua*
- *Tier: T={a,e,i,o,u}*
- *Erasing function yields: ioua*
- ***Grammar** that enforces vowel harmony: *ae, *ai, *ea, *io, etc.*
  → *this grammar rules out "bibobua"*

Tier-based languages over strings

Intuitive description:

- Project a tier
- Apply constraints over the tier

Tier-based Strictly Local languages

- Project a tier with the help of an erasing function – erase all nodes that are irrelevant for the constraint
- Apply SL constraints over the tier

Input-local tier-based Strictly Local Language (I-TSL)

- Project a tier with a *strictly local* function, i.e. nodes are projected with taking local context into consideration
- Apply SL constraints over the tier

Multiple I-TSL (MITSL)

- Project multiple tiers with a *strictly local* function
- Apply SL constraints over each tier (they can take different SL constraints)

Tier-based languages over trees

- Project a tree-tier from a tree
  - Simple erasing function in the case of TSL
  - ISL projection function in the case of I-TSL and MITSL
- Apply *substructure* constraints over the tree-tier (cf. Jardine (2016)), which equals to constraining the form of each node's daughter-string, based on that node's local context
  - *Example*: If `Merge` does not have negation as its sibling, then it cannot have NPI as its child.

We'll see more examples when we look at more NPI-licensing constraints.

Known results about subregular derivation trees

- Merge constraints are SL
- Merge with recursive adjunction is I-TSL (Graf, 2018)
- Move is I-TSL (Graf, 2018)
- C-command is not TSL (Vu, 2018) → ∃-NPI licensing is not TSL

→ Are NPI-licensing constraints in the quantifier-based approach I-TSL?

Outline

Licensing ∃-NPIs

- They must be c-commanded by negation at LF
- Two kinds of c-command relations:
  - Base c-command: movement does not play a role, nodes c-command each other in their base position
  - Derived c-command: movement plays a role, it either creates or destroys c-command relations

As it turns out, the two are different in terms of complexity.

Base c-command

Claim: *Base c-command is I-TSL.*

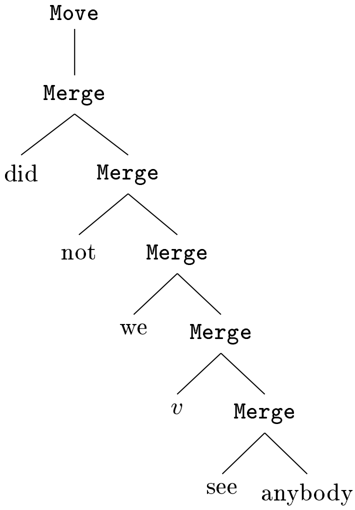
I show this on four examples:

Base c-command

Claim: *Base c-command is I-TSL.*

I show this on four examples:

- Negation base c-commands an NPI, and licenses it

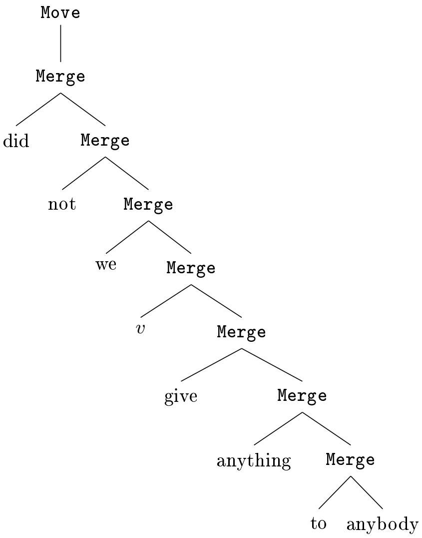


Base c-command

Claim: *Base c-command is I-TSL.*

I show this on four examples:

- Negation base c-commands an NPI, and licenses it
- Negation base c-commands multiple NPIs

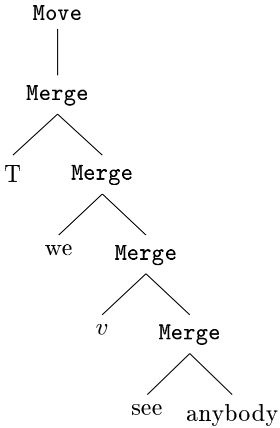


Base c-command

Claim: *Base c-command is I-TSL.*

I show this on four examples:

- Negation base c-commands an NPI, and licenses it
- Negation base c-commands multiple NPIs
- There is no negation to license the NPIs

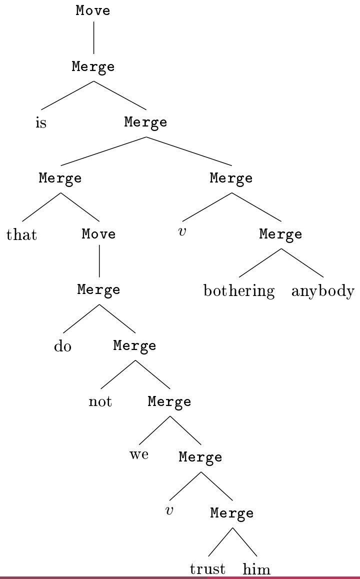


Base c-command

Claim: *Base c-command is I-TSL.*

I show this on four examples:

- Negation base c-commands an NPI, and licenses it
- Negation base c-commands multiple NPIs
- There is no negation to license the NPIs
- Negation does not c-command the NPIs



Projecting the tier based on local context

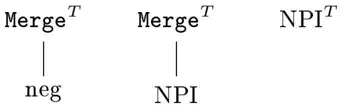
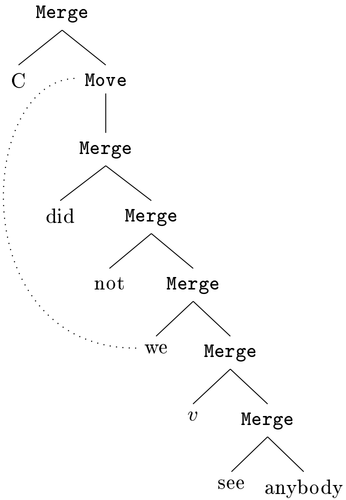
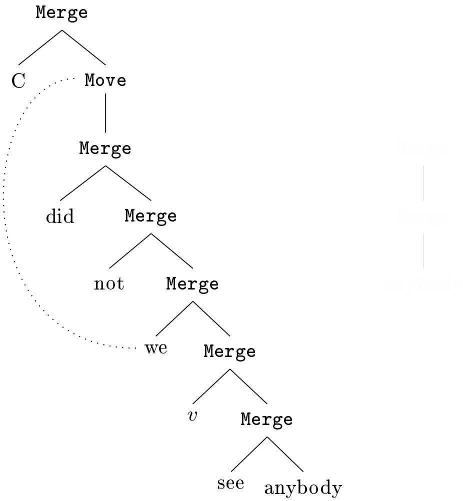


Figure 3: Contexts for the tier projection for English NPI-licensing



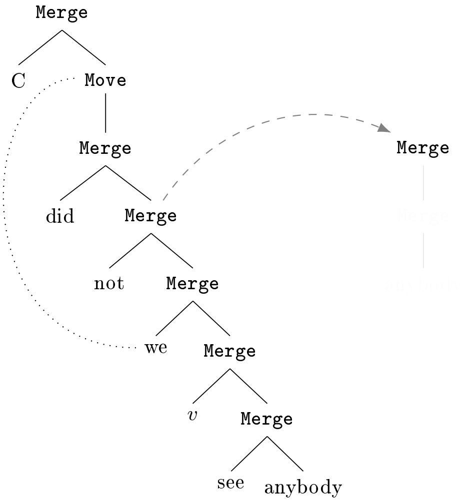
Projecting the tier based on local context

Figure 3: Contexts for the tier projection for English NPI-licensing



Projecting the tier based on local context

Figure 3: Contexts for the tier projection for English NPI-licensing



Projecting the tier based on local context

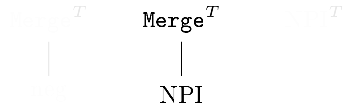
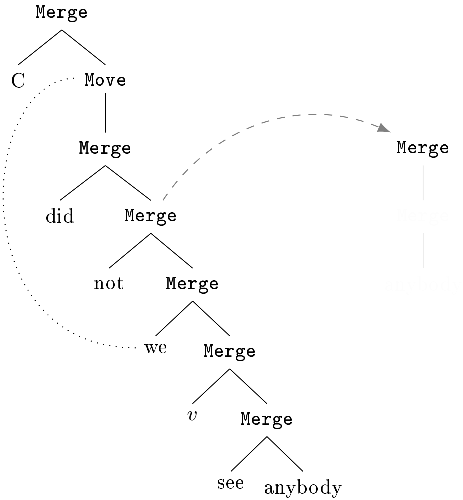


Figure 3: Contexts for the tier projection for English NPI-licensing



Projecting the tier based on local context

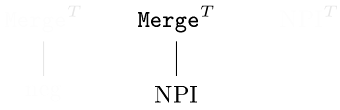
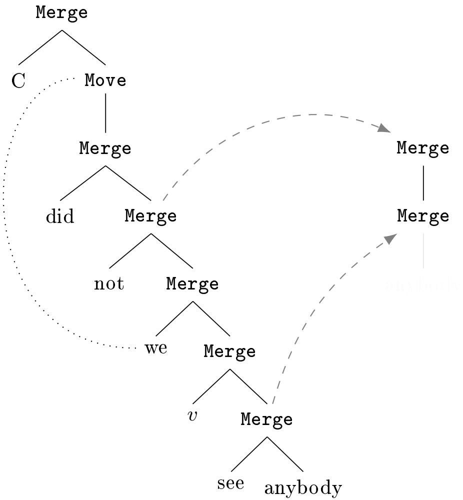
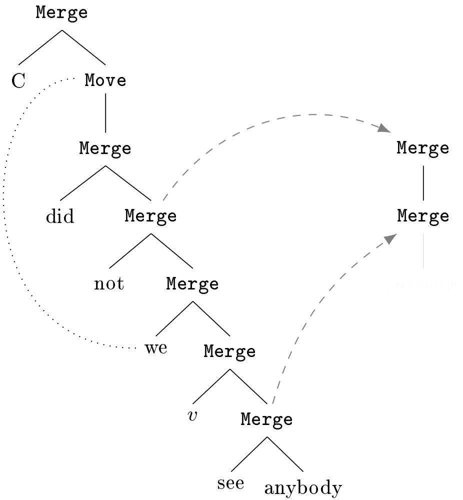


Figure 3: Contexts for the tier projection for English NPI-licensing



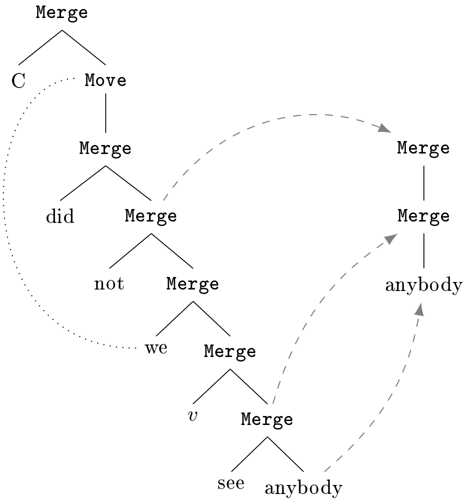
Projecting the tier based on local context

Figure 3: Contexts for the tier projection for English NPI-licensing



Projecting the tier based on local context

Figure 3: Contexts for the tier projection for English NPI-licensing



Applying SL constraints over the tier

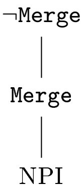


Figure 4: Banned substructure for English NPI-licensing, base c-command

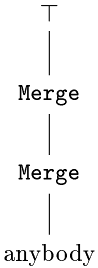


Figure 5: Projected tree-tier

Technically: If `Merge` has a non-`Merge` parent, then it cannot have an NPI among its children.

Applying SL constraints over the tier

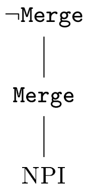


Figure 4: Banned substructure for English NPI-licensing, base c-command

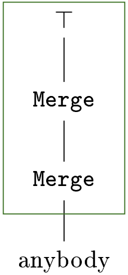


Figure 5: Projected tree-tier

Technically: If Merge has a non-Merge parent, then it cannot have an NPI among its children.

Applying SL constraints over the tier

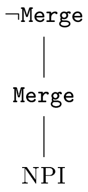


Figure 4: Banned substructure for English NPI-licensing, base c-command

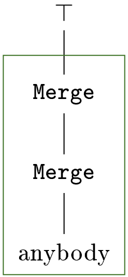


Figure 5: Projected tree-tier

Technically: If `Merge` has a non-`Merge` parent, then it cannot have an NPI among its children.

Applying SL constraints over the tier

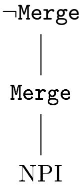


Figure 4: Banned substructure for English NPI-licensing, base c-command

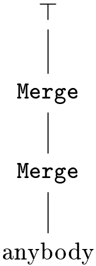


Figure 5: Projected tree-tier

Technically: If `Merge` has a non-`Merge` parent, then it cannot have an NPI among its children.

→ This tree-tier does not violate the SL constraint in Figure 2.

Licensing multiple NPIs

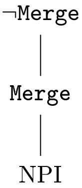
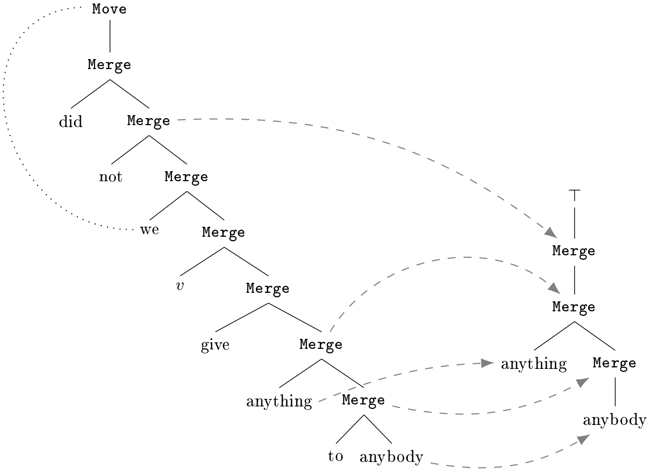


Figure 6: Banned substructure



Licensing multiple NPIs

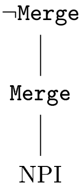
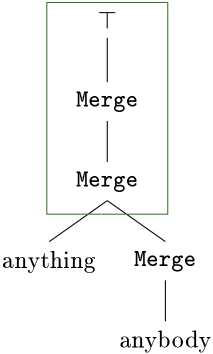


Figure 6: Banned substructure



Licensing multiple NPIs

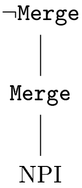
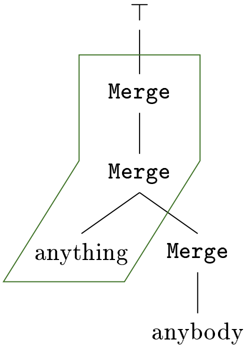


Figure 6: Banned substructure



Licensing multiple NPIs

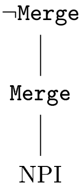
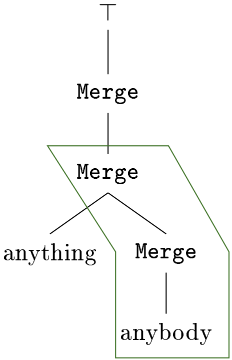


Figure 6: Banned substructure



Licensing multiple NPIs

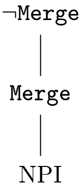
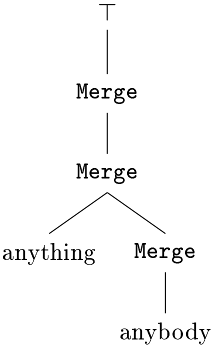


Figure 6: Banned substructure



→ This tree-tier does not violate the SL constraint in Figure 4.

Ruling out unlicensed constructions

1. There is no negation in the sentence:

   (37) * We saw anybody.

2. Negation does not c-command the NPI

   (38) * That we do not trust him is bothering anyone.

No licensor

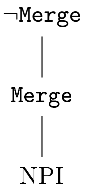
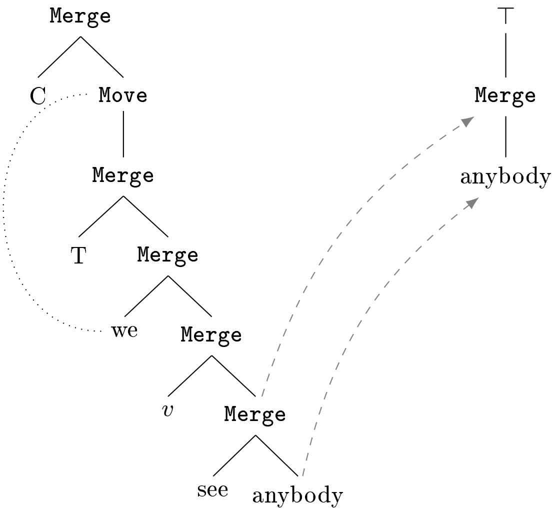


Figure 7: Banned substructure



No licensor

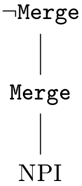
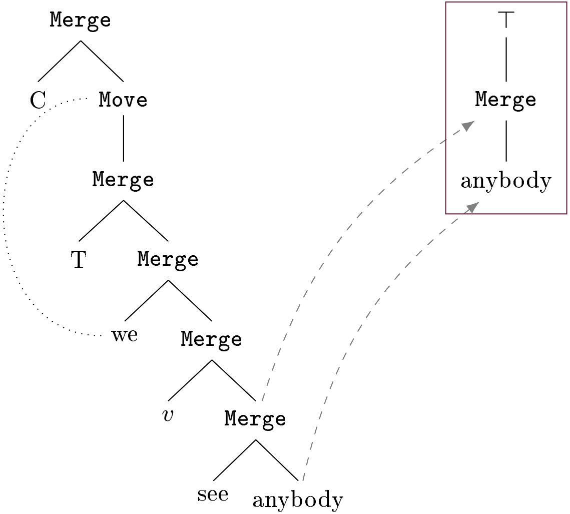


Figure 7: Banned substructure



No licensor

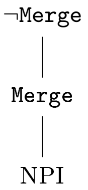
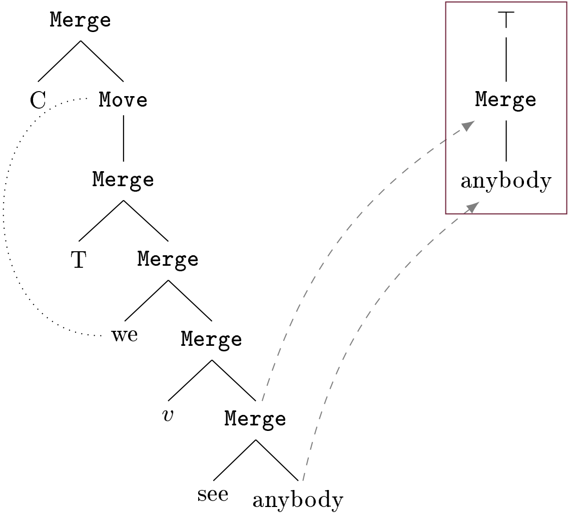


Figure 7: Banned substructure



→ This tree-tier violates the SL constraint in Figure 5.

No c-commanding licensor

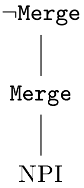
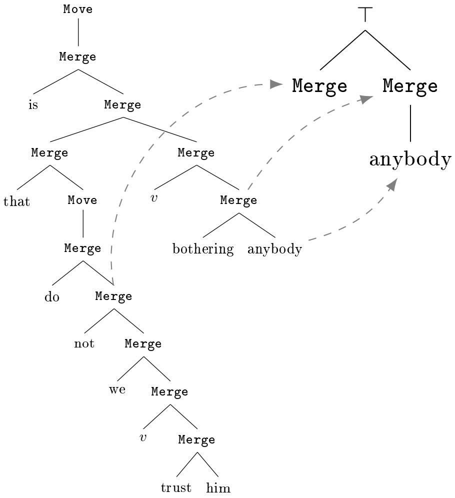


Figure 8: Banned substructure



No c-commanding licensor

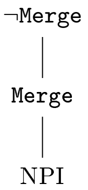
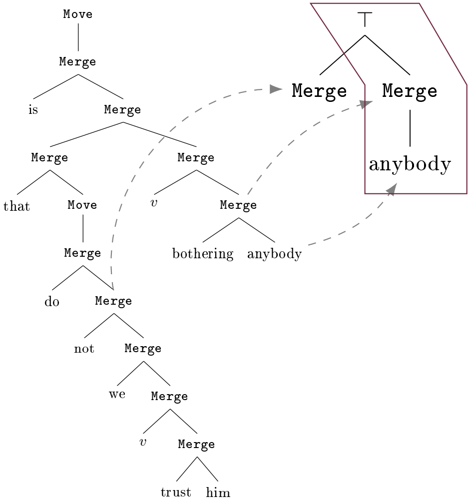


Figure 8: Banned substructure



No c-commanding licensor

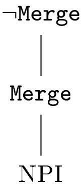
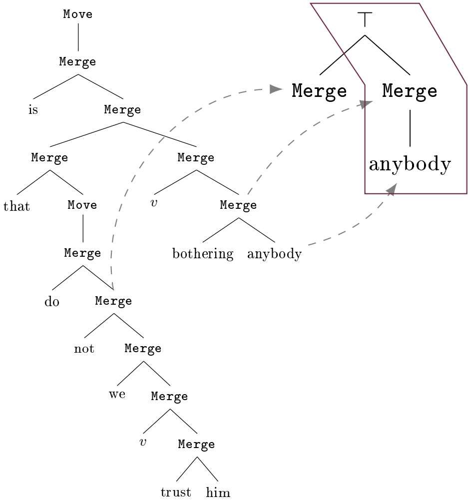


Figure 8: Banned substructure



→ This tree-tier violates the SL constraint in Figure 6.

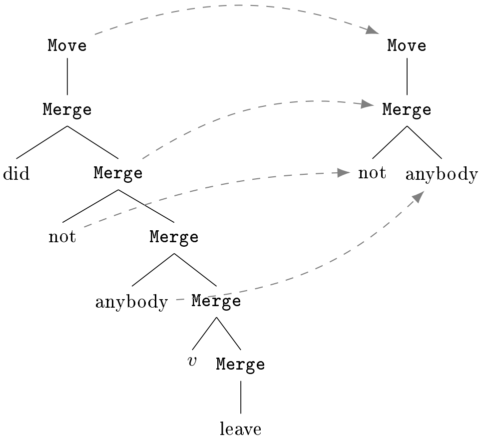
Derived c-command

Claim: *Derived c-command is not I-TSL.*

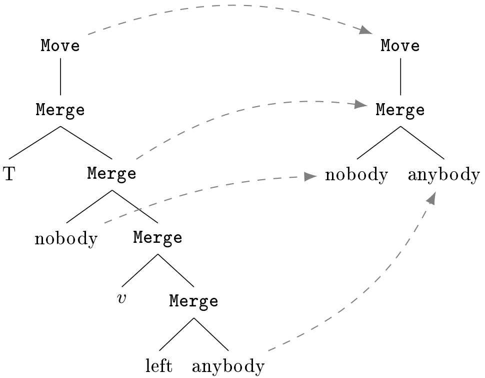
- To determine if a moved node x c-commands another node, we need to project the `Move` node associated with x
- Because of the long-distance nature of Move, there is no function that can project the right Move node based on *local* context
- Even if there is a function that can, tree-tiers projected from grammatical and ungrammatical sentences can be indistinguishable

Derived c-command

(39) * Anybody did not leave.



(40) Nobody left anybody.



Interim summary

- Base c-command can be described in terms of I-TSL
- Derived c-command cannot be described in terms of I-TSL

Outline

Licensing ∀-NPIs

Recap of the licensing mechanism for ∀-NPIs:

- NPI must scope higher than negation
- To achieve this, NPI undergoes QR (either overt or covert) to NegP
- Covert QR is clause-bounded, overt QR is not

How to model this?

Licensing ∀-NPIs

Recap of the licensing mechanism for ∀-NPIs:

- NPI must scope higher than negation
- To achieve this, NPI undergoes QR (either overt or covert) to NegP
- Covert QR is clause-bounded, overt QR is not

How to model this?

- The first part of the licensing mechanisms looks like reverse ∃-NPI licensing – now NPI has to c-command negation
  - This would yield the same complexity results as for ∃-NPIs: base c-command is I-TSL, derived c-command is not
  - It does not get to the other two points

Licensing ∀-NPIs

Recap of the licensing mechanism for ∀-NPIs:

- NPI must scope higher than negation
- To achieve this, NPI undergoes QR (either overt or covert) to NegP
- Covert QR is clause-bounded, overt QR is not

How to model this?

- The first part of the licensing mechanisms looks like reverse ∃-NPI licensing – now NPI has to c-command negation
  - This would yield the same complexity results as for ∃-NPIs: base c-command is I-TSL, derived c-command is not
  - It does not get to the other two points
- If we assume that *all* ∀-NPIs always undergo movement, we can just state the constraints as as move and locality constraints
  - Both can be captured with I-TSL constraints

Licensing ∀-NPIs

Recap of the licensing mechanism for ∀-NPIs:

- NPI must scope higher than negation
- To achieve this, NPI undergoes QR (either overt or covert) to NegP
- Covert QR is clause-bounded, overt QR is not

How to model this?

- The first part of the licensing mechanisms looks like reverse ∃-NPI licensing – now NPI has to c-command negation
  - This would yield the same complexity results as for ∃-NPIs: base c-command is I-TSL, derived c-command is not
  - It does not get to the other two points
- If we assume that *all* ∀-NPIs always undergo movement, we can just state the constraints as as move and locality constraints
  - Both can be captured with I-TSL constraints

To keep this discussion simple, I only show licensing of a single NPI. For licensing multiple NPIs, we will need to use `Cluster`, but `Cluster` constraints are also I-TSL.

Assumed lexicon

- The NPI always moves → I stipulate that movement is triggered by a −npi movement feature
  - −npi for overt movement
  - $-_s$npi for covert movement
- The NPI moves to NegP → negation must be able to have a +npi feature to license movement
  - +npi for overt movement
  - $+_s$npi for covert movement

	Move licensee	Move licensor
Overt Move	NPI :: d −npi	nem :: =t +npi t
Covert Move	NPI :: d $-_s$npi	nem :: =t $+_s$npi t

Tier-projections

Project two tiers:

- Move-tier

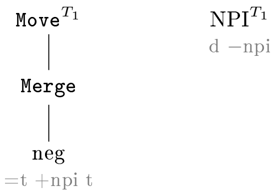
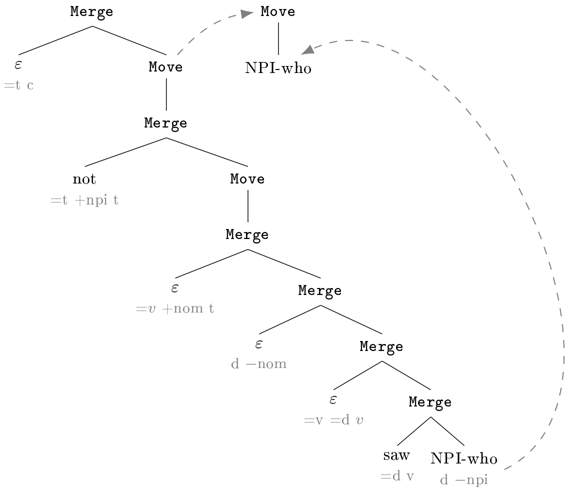


Figure 9: Contexts for the **Move** tier



(41) Sen-ki-t nem lát-t-am.
NPI-who-ACC NEG see-PST-1SG
'I did not see anyone.'

Tier-projections

Project two tiers:

- Move-tier
- S-move-tier

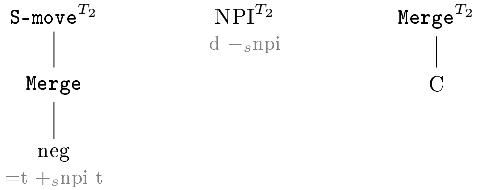
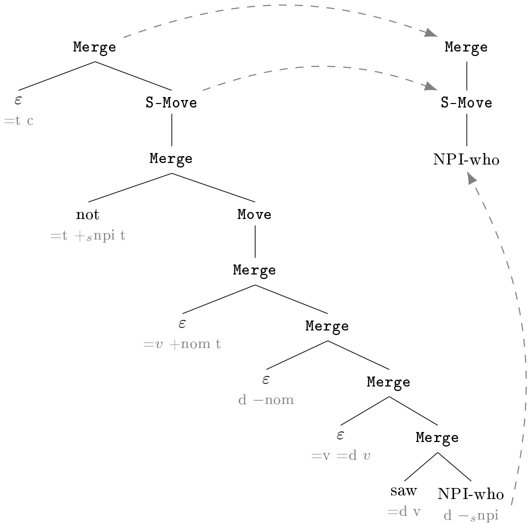


Figure 10: Contexts for the **S-move** tier

(44) Nem lát-t-am sen-ki-t.
NEG see-PST-1SG NPI-who-ACC
'I did not see anyone.'



Constraints on the Move-tier

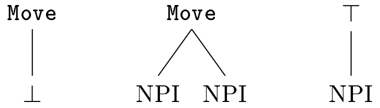


Figure 11: Banned substructures for the `Move` tier

Technically: `Move` must have exactly one NPI-child.

Constraints on the Move-tier

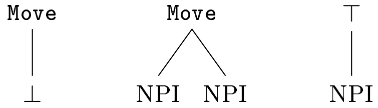
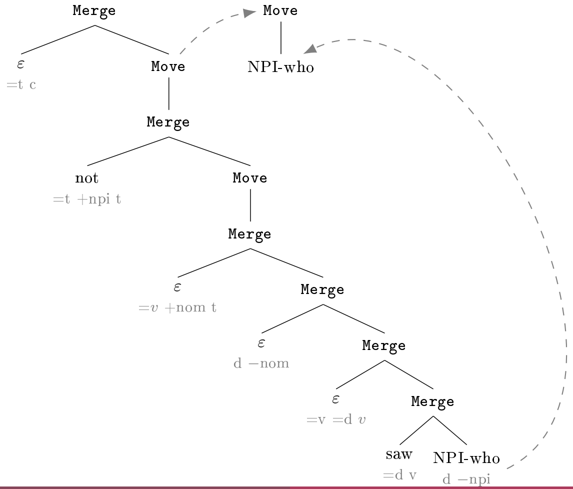


Figure 11: Banned substructures for the Move tier

Technically: Move must have exactly one NPI-child.

(47) Sen-ki-t nem lát-t-am.
NPI-who-ACC NEG see-PST-1SG
'I did not see anyone.'



Constraints on the Move-tier

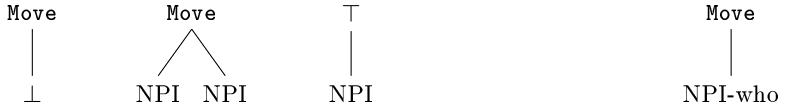
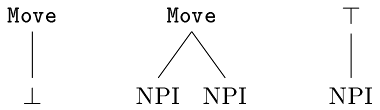


Figure 11: Banned substructures for the Move tier

Technically: Move must have exactly one NPI-child.

(49) Sen-ki-t nem lát-t-am.
NPI-who-ACC NEG see-PST-1SG
'I did not see anyone.'

Constraints on the Move-tier



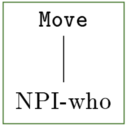


Figure 11: Banned substructures for the Move tier

Technically: Move must have exactly one NPI-child.

(51) Sen-ki-t nem lát-t-am.
NPI-who-ACC NEG see-PST-1SG
'I did not see anyone.'

Constraints on the Move-tier

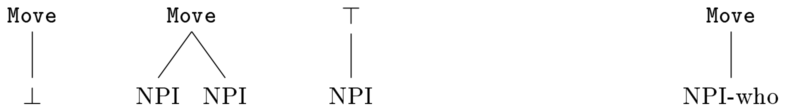


Figure 11: Banned substructures for the Move tier

Technically: Move must have exactly one NPI-child.

(53) Sen-ki-t nem lát-t-am.
NPI-who-ACC NEG see-PST-1SG
'I did not see anyone.'

→ The tier-tree does not violate any of the constraints in Figure 11.

Constraints on the Move-tier

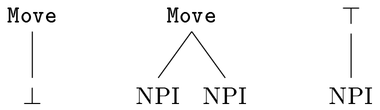
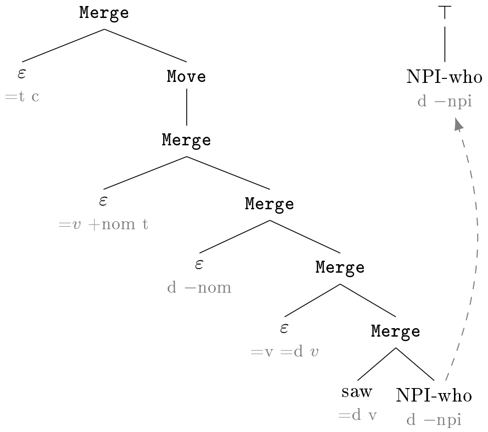


Figure 11: Banned substructures for the `Move` tier

Technically: `Move` must have exactly one NPI-child.

(56) * Lát-t-am sen-ki-t.
see-PST-1SG NPI-who-ACC



Constraints on the Move-tier

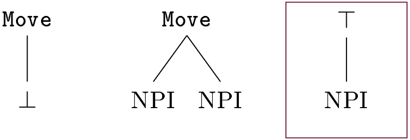
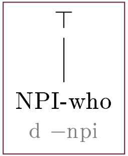


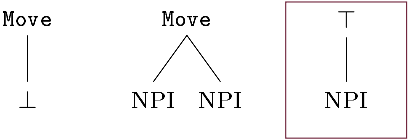
Figure 11: Banned substructures for the `Move` tier



Technically: `Move` must have exactly one NPI-child.

(58) * Lát-t-am sen-ki-t.
see-PST-1SG NPI-who-ACC

Constraints on the Move-tier



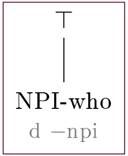


Figure 11: Banned substructures for the `Move` tier

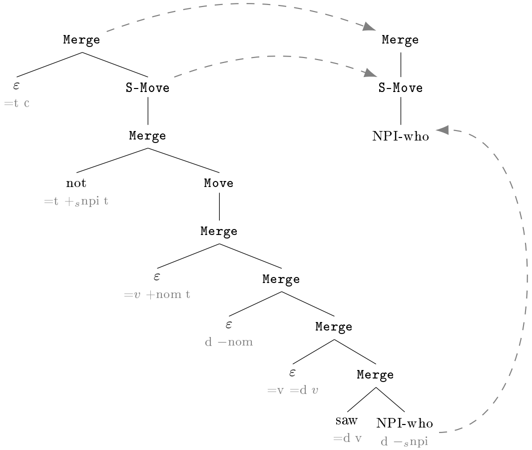
Technically: `Move` must have exactly one NPI-child.

(60) * Lát-t-am sen-ki-t.
see-PST-1SG NPI-who-ACC

→ The tier-tree violates one of the constraints in Figure 11.

Move constraints on the S-move-tier

(61) Nem lát-t-am sen-ki-t.
NEG see-PST-1SG NPI-who-ACC
'I did not see anyone.'



Move constraints on the S-move-tier

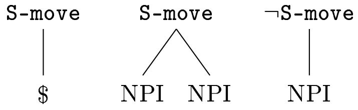
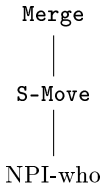


Figure 12: Banned substructures for the `S-move` tier

(64) Nem lát-t-am sen-ki-t.
NEG see-PST-1SG NPI-who-ACC
'I did not see anyone.'



Move constraints on the S-move-tier

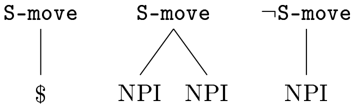
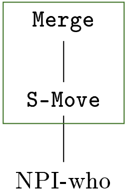


Figure 12: Banned substructures for the S-move tier

(66) Nem lát-t-am sen-ki-t.
NEG see-PST-1SG NPI-who-ACC
'I did not see anyone.'



Move constraints on the S-move-tier

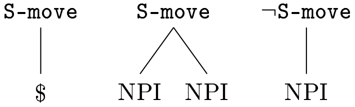
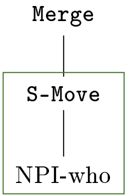


Figure 12: Banned substructures for the S-move tier

(68) Nem lát-t-am sen-ki-t.
NEG see-PST-1SG NPI-who-ACC
'I did not see anyone.'



Move constraints on the S-move-tier

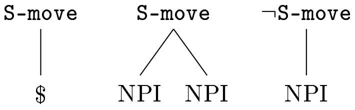
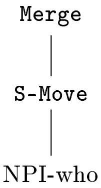


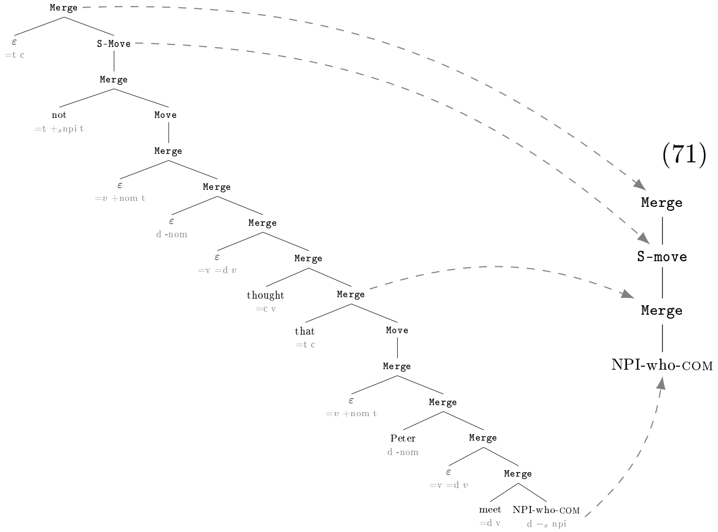
Figure 12: Banned substructures for the `S-move` tier



(70) Nem lát-t-am sen-ki-t.
NEG see-PST-1SG NPI-who-ACC
'I did not see anyone.'

→ None of the constraints are violated in the tier-tree.

Locality constraints on the S-move tier



(71) * Nem gondol-t-am, hogy Péter találkoz-na sen-ki-vel.
NEG think-PST-1SG that Peter meet-COND.3SG

‘I did not think that Peter would meet with anyone.’

Locality constraints on the S-move tier

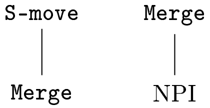
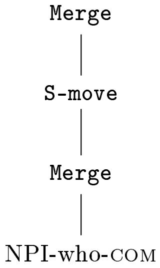


Figure 13: Banned substructures for the `S-move` tier

(74) * Nem gondol-t-am, hogy Péter találkoz-na sen-ki-vel.
NEG think-PST-1SG that Peter meet-COND.3SG
'I did not think that Peter would meet with anyone.'



Locality constraints on the S-move tier

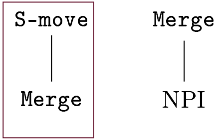
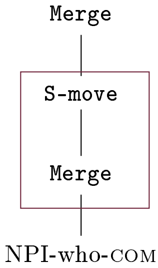


Figure 13: Banned substructures for the `S-move` tier

(76) * Nem gondol-t-am, hogy Péter
NEG think-PST-1SG that Peter
találkoz-na sen-ki-vel.
meet-COND.3SG
'I did not think that Peter would meet with anyone.'



Locality constraints on the S-move tier

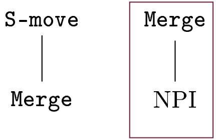
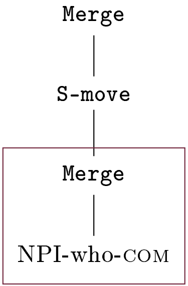


Figure 13: Banned substructures for the `S-move` tier

(78) * Nem gondol-t-am, hogy Péter találkoz-na sen-ki-vel.
NEG think-PST-1SG that Peter meet-COND.3SG
'I did not think that Peter would meet with anyone.'



Locality constraints on the S-move tier

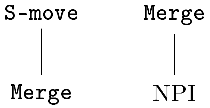
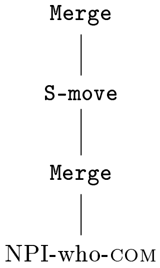


Figure 13: Banned substructures for the `S-move` tier

(80) * Nem gondol-t-am, hogy Péter találkoz-na sen-ki-vel.
NEG think-PST-1SG that Peter meet-COND.3SG
'I did not think that Peter would meet with anyone.'



→ This tier-tree violates both of the locality constraints.

Outline

Summary

- ∃-**NPIs**: licensor must c-command the NPI at LF
  - Base c-command is an Input Local-TSL constraint (I-TSL)
  - Derived c-command is *not* I-TSL
- ∀-**NPIs**: NPI must c-command the licensor at LF (achieved through Quantifier-raising (QR))
  - If we stipulate all the necessary features to ensure that NPIs *always* move to NegP at LF, then we only need constraints to regulate `Move` and `S-move` (which are needed for well-formed derivation trees also)
  - We had to project two separate tiers, making the overall NPI-licensing constraint Multiple Input-local Tier-based Strictly Local (MITSL)

Take-away

Theoretical analysis can help lower the computational complexity of syntactic constraints.

- Assuming a hierarchical structure rather than a string as the relevant data structure *lowers* the power of the necessary logic to describe syntactic dependencies
- Assuming covert movement as a core mechanism in the licensing of universally quantified NPIs *lowers* the computational complexity of their constraints

Implications

If all syntactic dependencies are *subregular*, then

- We have a better grasp on possible and impossible typological patterns
- We can develop more effective learning algorithms
- We can develop more effective processing algorithms

What's next?

We are still in the beginning of studying subregular syntax. There is a lot to do!

- Give a similar analysis of other NPI-licensing aproaches, e.g. Collins and Postal's (2015) NEG-raising account which assumes movement for ∃-NPIs as well
- Pin-down the complexity of derived c-command dependencies
- Develop learning algorithms for subregular tree-languages
- Map out other syntactic dependencies
- Study the nature of the mapping functions from derivation trees to outputs
- Study different representations of syntactic derivation, e.g. dependency trees (Graf and De Santo, 2019)

Thank you for coming!

References I

Chandlee, J. (2014). Strictly local phonological processes. Ph. D. thesis, University of Delaware.

Collins, C. and P. M. Postal (2015). A Typology of Negative Polarity Items.

Gärtner, H.-m. and J. Michaelis (2010). On the Treatment of Multiple-Wh-Interrogatives in Minimalist Grammars. In T. Hanneforth and G. Fanselow (Eds.), Language and Logos, pp. 339–366. Berlin: Akademie Verlag.

Giannakidou, A. (2000). Negative... Concord? Natural Language & Linguistic Theory& Linguistic Theory 18(3), 457–523.

Graf, T. (2018). Why movement comes for free once you have adjunction. Proceedings of CLS 53, 117–137.

Graf, T. and A. De Santo (2019). Sensing Tree Automata as a Model of Syntactic Dependencies. Proceedings of the 16th Meeting on the Mathematics of Language (MOL 2019).

Graf, T., A. De Santo, J. Rawski, A. Aksenova, H. Dolatian, S. Moradi, H. Baek, S. Yang, and J. Heinz (2018). Tiers and Relativized Locality Across Language Modules.

Graf, T. and J. Heinz (2015). Commonality in Disparity : The Computational View of Syntax and Phonology A New View of the Power of Syntax and Phonology.

Grewendorf, G. (2001). Multiple Wh-Fronting. Linguistic Inquiry 32(1), 87–122.

Jardine, A. (2016). Locality and non-linear representations in tonal phonology. Ph. D. thesis, University of Delaware.

References II

Joshi, A. K. (1985). Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In D. Dowty, Karttuhen, and A. Zwicky (Eds.), Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives, pp. 206–250. Cambridge University Press.

Kobele, G. M., C. Retoré, and S. Salvati (2007). An automata-theoretic approach to minimalism. Model theoretic syntax at 10, 71–80.

Morawietz, F. (2003). Two-Step Approaches to Natural Language Formalism, Volume 64. New York: Mouton de Gruyter.

Reinhart, T. (1976). The syntactic domain of anaphora. Ph. D. thesis, Massachussetts Institute of Technology.

Sabel, J. (2001). Deriving Multiple Head and Phrasal Movement: The Cluster Hypothesis. Linguistic Inquiry 32(3), 532–547.

Shieber, S. M. (1985). Evidence against the context-freeness of natural language. Linguistics and Philosophy 8, 333–343.

Stabler, E. (1997). Derivational minimalism. In Logical aspects of computational linguistics, pp. 68–95.

Thatcher, J. W. (1967). Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. Journal of Computer and System Sciences 1(4), 317–322.

Vu, M. H. (2018). Towards a formal description of NPI-licensing patterns. Poster presentation at the Society of Computation in Language, Salt Lake City, UT.

Wurmbrand, S. (2018). The cost of raising quantifiers. Glossa: a journal of general linguistics 3(1), 1–40.