# Managing scope ambiguities in Glue via multistage proving

Jamie Y. Findlay and Dag T. T. Haug
jamie.findlay@iln.uio.no
d.t.t.haug@ifikk.uio.no

University of Oslo

27th International Lexical-Functional Grammar Conference

12 July 2022

# Overview

- Glue Semantics overgenerates when it comes to scope ambiguities.
- This has both theoretical and practical implications.
- We propose a lightweight, modular solution – a new level of the projection architecture which can limit scopal interactions.
  - No changes to the linear logic or proof algorithm.
  - In keeping with the LFG philosophy.
- We demonstrate its success on three problems for Glue regarding scope.

# Outline

# The problems

# Three scope problems

1. Modifier scope:

(1) an alleged former racketeer (Andrews & Manning 1999)
$\Rightarrow$ **alleged(former(racketeer))**
$\not\Rightarrow$ **former(alleged(racketeer))**

(2) a counterfeit American coin (Campbell 2002)
$\Rightarrow$ **counterfeit(american(coin))**
$\not\Rightarrow$ **american(counterfeit(coin))**

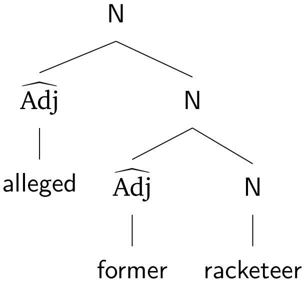
(3) a trustworthy former chairman (Lowe 2015: 441)
$\Rightarrow$ **trustworthy(former(chairman))**
$\not\Rightarrow$ **former(trustworthy(chairman))**

(4) a trustworthy Scottish chairman
$\Rightarrow$ **trustworthy(scottish(chairman))**
$\Rightarrow$ **scottish(trustworthy(chairman))**

(**trustworthy(scottish(chairman))** $\leftrightarrow$ **scottish(trustworthy(chairman))**)

# Modifier scope

- Traditional (non-LFG) solution: semantic scope follows from c-structure scope.
    - But this is not possible 'out of the box' in LFG+Glue, since f-structure flattens this relationship.




$$r\left[\begin{array}{ll}\text{PRED} & \text{`racketeer'}\\ \text{ADJ} & \left\{\begin{array}{l}\left[\text{PRED}\quad \text{`alleged'}\right]\\ \left[\text{PRED}\quad \text{`former'}\right]\end{array}\right\}\end{array}\right]$$

$$\begin{array}{l}\textbf{racketeer} : (e_r \multimap t_r)\\ \textbf{alleged} : (e_r \multimap t_r) \multimap (e_r \multimap t_r)\\ \textbf{former} : (e_r \multimap t_r) \multimap (e_r \multimap t_r)\end{array} \Rightarrow \begin{array}{l}\textbf{former}(\textbf{alleged}(\textbf{racketeer}))\\ \textbf{alleged}(\textbf{former}(\textbf{racketeer}))\end{array}$$

# Three scope problems

2. Scope islands:

(5) **An accomplice** ensured [that **every prisoner** escaped]. (Gotham 2021: 151)

a. $=$ There is an accomplice who ensured that every prisoner escaped. $(\exists > \forall)$

b. $=$ For every prisoner, there is an accomplice who ensured they escaped. $(\forall > \exists)$

(6) **A warden** thinks [that **every prisoner** escaped]. (Gotham 2021: 150)

a. $=$ There is a warden who thinks that every prisoner escaped. $(\exists > \forall)$

b. $\neq$ For every prisoner, there is a warden who thinks they escaped. $({}^{*}\forall > \exists)$

# Scope islands

- Not just determined by syntactic structure (tree topology or clause boundaries).
- Rather, poorly understand interactions between embedding operators and the quantifiers they embed (Barker 2021).
- Gotham (2021) provides a Glue-based analysis, but at the cost of complexifying the linear logic component.

# Three scope problems

3. Sublexical meanings:

- Recent work in Glue[1] has proposed to break down lexical meaning (using TEMPLATES) so that common parts can be shared across lexical entries.

(7) crushed V $(\uparrow \textsc{pred}) = \text{`crush'}$ (after Asudeh et al. 2014)

$@\textsc{Agent}$

$@\textsc{Patient}$

$@\textsc{Passive}$

$\lambda e.\mathbf{crush}(e) : v_{\uparrow} \multimap t_{\uparrow}$

[1](E.g. Asudeh et al. 2014; Przepiórkowski 2017; Findlay 2020)

# Sublexical meanings

(8) $\lambda e.\mathbf{crush}(e) : v_{\uparrow} \multimap t_{\uparrow}$

(9) $\textsc{Agent}$:=
$\lambda P.\lambda x.\lambda e.P(e) \wedge \mathbf{agent}(e, x) : (v_{\uparrow} \multimap t_{\uparrow}) \multimap e_{(\uparrow \mathrm{OBL}_{\mathrm{AG}})} \multimap v_{\uparrow} \multimap t_{\uparrow}$[2]

(10) $\textsc{Patient}$:=
$\lambda P.\lambda x.\lambda e.P(e) \wedge \mathbf{patient}(e, x) : (v_{\uparrow} \multimap t_{\uparrow}) \multimap e_{(\uparrow \mathrm{SUBJ})} \multimap v_{\uparrow} \multimap t_{\uparrow}$

(11) $\textsc{Passive}$:=
$(\uparrow \textsc{voice}) = \textsc{passive}$
$\Big(\lambda P.\exists x[P(x)] : (e_{(\uparrow \mathrm{OBL}_{\mathrm{AG}})} \multimap t_{\uparrow}) \multimap t_{\uparrow}\Big)$

[2] We simplify by resolving the mapping of $\textsc{arg}1$ and $\textsc{arg}2$ to their GFs in the passive.

# Sublexical meanings

- Asudeh et al. (2014: Fig. 7) give 1 proof from 7 meaning constructors for *Kim was crushed last night*.

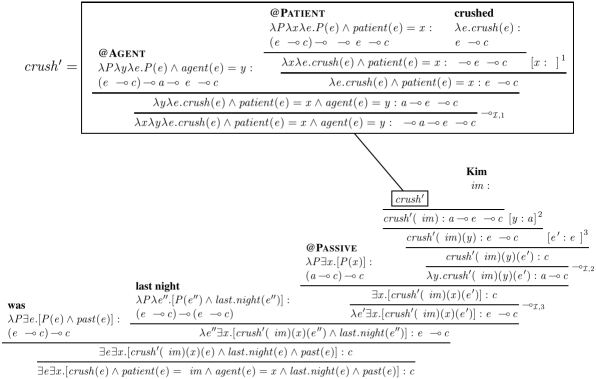



Figure 7: Proof for *Kim was crushed last night.*

- But in fact there are 20 distinct proofs from these premises(!), corresponding to only 1 reading.

# The proposal

# The proposal

- We want to control the order of composition in the Glue proof: some things should combine before others (they should be 'boxed off').
- Other proposals have done this via additions to the linear logic (Gotham 2019, 2021), or via constraints on the proof algorithm (Crouch & van Genabith 1999).
- Andrews (2018: 141f.) offers a solution by encoding c-structure information into f-structure – but this arguably breaks modularity.
- We propose a more LFG-style solution, by making use of an additional level of representation in the projection architecture.
    - This also means we can continue to use existing implementations for linear logic proving (Hepple 1996; Lev 2007; Messmer & Zymla 2018).

# Proof-structure

- We call this level of representation PROOF-STRUCTURE:

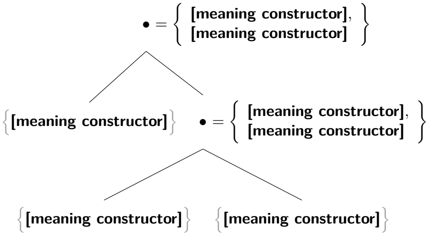
- A tree, where each sub-tree represents a sub-proof.
- Nodes are (sets of) premises (i.e. meaning constructors).
- One meaning constructor from each daughter node is used in the proof of their mother.
- Projected from c-structure via a function $\gamma$.
- Lexically-contributed meaning constructors are introduced as daughters of pre-terminal nodes:

(12) $\hat{*}_\gamma \lhd$ **[meaning constructor]**

- In the default cause, phrase-structure rules produce a flat proof-structure (the vanilla LFG+Glue mode):

(13) $$\mathrm{S} \rightarrow \begin{array}{cc} \mathrm{NP} & \mathrm{VP} \\ (\uparrow \textsc{subj}) = \downarrow & \uparrow = \downarrow \\ \hat{*}_\gamma = *_\gamma & \hat{*}_\gamma = *_\gamma \end{array}$$

- But this can be selectively interrupted!



# Modifier scope

- We annotate the pre-nominal adjective rule of English as follows:

(14)
$$\mathsf{N} \rightarrow \quad \underset{\substack{\downarrow \in (\uparrow \text{ ADJ}) \\ \hat{*}_\gamma = *_\gamma}}{\widehat{\text{Adj}}} \qquad \underset{\substack{\uparrow = \downarrow \\ \hat{*}_\gamma \lhd *_\gamma}}{\mathsf{N}}$$

$\Rightarrow$ Meaning for the nominal head is its own sub-proof, fixing a scope ordering based on c-structure.

# Modifier scope: lexical entries

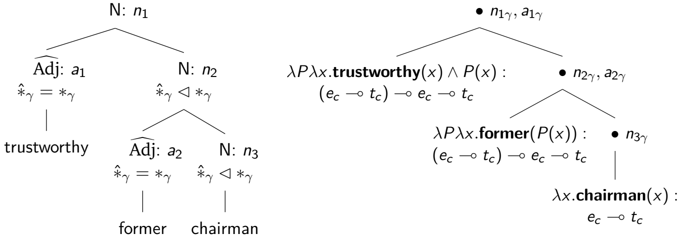
(15) former $\widehat{\text{Adj}}$
$(\uparrow \textsc{pred}) = \text{`former'}$
$\%N = (\textsc{adj} \in \uparrow)$
$\hat{*}_\gamma \lhd \lambda P \lambda x.\textbf{former}(P(x)) : (e_{\%N} \multimap t_{\%N}) \multimap e_{\%N} \multimap t_{\%N}$

(16) trustworthy $\widehat{\text{Adj}}$
$(\uparrow \textsc{pred}) = \text{`trustworthy'}$
$\%N = (\textsc{adj} \in \uparrow)$
$\hat{*}_\gamma \lhd \lambda P \lambda x.\textbf{trustworthy}(x) \wedge P(x) : (e_{\%N} \multimap t_{\%N}) \multimap e_{\%N} \multimap t_{\%N}$

(17) chairman N
$(\uparrow \textsc{pred}) = \text{`chairman'}$
$\hat{*}_\gamma \lhd \lambda x.\textbf{chairman}(x) : e_{\uparrow} \multimap t_{\uparrow}$

# Modifier scope: structures

(18)

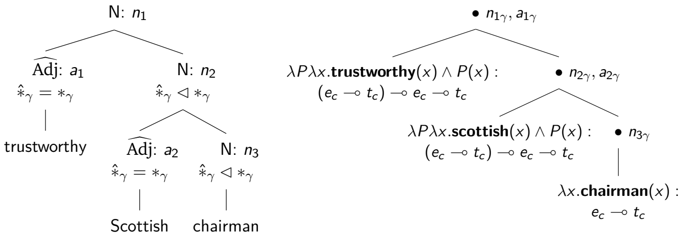



$\Rightarrow$ **trustworthy(former(chairman))**

$\not\Rightarrow$ **former(trustworthy(chairman))**

# Modifier scope: no spurious ambiguity

(19)




$\Rightarrow$ **trustworthy(scottish(chairman))**

$\not\Rightarrow$ **scottish(trustworthy(chairman))**

# Scope islands

- Reminder:

(20) **An accomplice** ensured [that **every prisoner** escaped].

a. = There is an accomplice who ensured that every prisoner escaped. ($\exists > \forall$)

b. = For every prisoner, there is an accomplice who ensured they escaped. ($\forall > \exists$)

(21) **A warden** thinks [that **every prisoner** escaped].

a. = There is a warden who thinks that every prisoner escaped. ($\exists > \forall$)

b. ≠ For every prisoner, there is a warden who thinks they escaped. ($^*\forall > \exists$)

# Scope islands

- Quantifiers cannot scope outside of sub-proofs, since we require that sub-proofs be complete proofs, i.e. that all hypotheses are discharged.
  - So sub-trees in proof-structure are scope islands.

(22)
$$\mathrm{V}' \rightarrow \quad \begin{array}{c} \mathrm{V} \\ \uparrow = \downarrow \\ \hat{*}_\gamma = *_\gamma \end{array} \quad \begin{array}{c} \mathrm{CP} \\ (\uparrow \textsc{comp}) = \downarrow \\ \hat{*}_\gamma \lhd *_\gamma \end{array}$$

# Scope islands

- $\bullet = \mathbf{a}(y, \mathbf{warden}(y), \mathbf{think}(y, \mathbf{every}(x, \mathbf{prisoner}(x), \mathbf{escape}(x)))) : t_t$
  - $\lambda R.\lambda S.\mathbf{a}(y, R(y), S(y)) :$ $(e_w \multimap t_w) \multimap \forall H[(e_w \multimap t_H) \multimap t_H]$
  - $\lambda x.\mathbf{warden}(x) :$ $e_w \multimap t_w$
  - $\lambda p.\lambda x.\mathbf{think}(x, p) :$ $t_e \multimap e_w \multimap t_t$
  - $\bullet = \mathbf{every}(x, \mathbf{prisoner}(x), \mathbf{escape}(x)) : t_e$
    - $\lambda R.\lambda S.\mathbf{every}(x, R(x), S(x)) :$ $(e_p \multimap t_p) \multimap \forall H[(e_p \multimap t_H) \multimap t_H]$
    - $\lambda x.\mathbf{prisoner}(x) :$ $e_p \multimap t_p$
    - $\lambda x.\mathbf{escape}(x) :$ $e_p \multimap t_e$

# Scope islands

- Whether such sub-trees are added or not can depend on other features:

(23)
$$\mathrm{V}' \rightarrow \quad \begin{array}{c} \mathrm{V} \\ \uparrow = \downarrow \\ \hat{*}_\gamma = *_\gamma \end{array} \quad \begin{array}{c} \mathrm{CP} \\ (\uparrow \textsc{comp}) = \downarrow \\ \left\{ \hat{*}_\gamma = *_\gamma \;\middle|\; \begin{array}{c} \hat{*}_\gamma \lhd *_\gamma \\ (\downarrow_\sigma \textsc{ScopeIsland}) =_c + \end{array} \right\} \end{array}$$

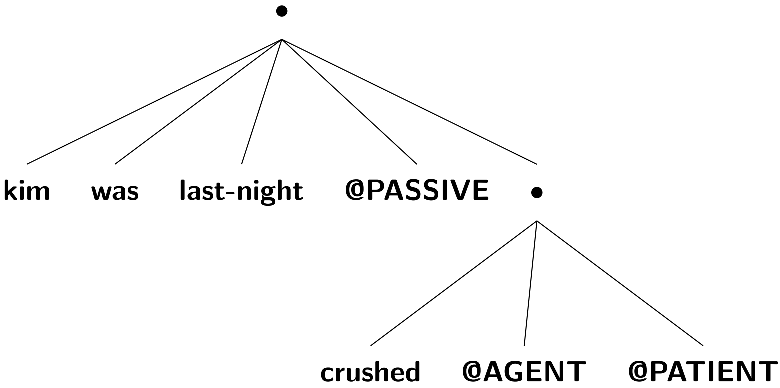
# Sublexical meanings

- To isolate sublexical meanings as a sub-proof, we make reference to the terminal node in the c-structure (which is then explicitly connected to the rest of the proof-structure tree).
- Sublexical meaning constructors are then made daughters of $*_\gamma$ rather than of $\hat{*}_\gamma$.

(24) crushed V
$(\uparrow \text{PRED}) = \text{`crush'}$
$\hat{*}_\gamma \lhd *_\gamma$
$*_\gamma \lhd \lambda e.\textbf{crush}(e) : v_\uparrow \multimap t_\uparrow$
@PASSIVE & @AGENT & @PATIENT

# Sublexical meanings

(25)




- Reduces number of proofs from 20 to 3.
  - (or 1 if **@PASSIVE** is also included)

# Sublexical meanings

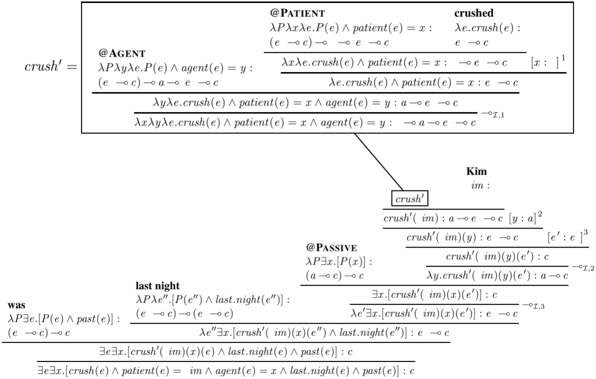



Figure 7: Proof for *Kim was crushed last night.*

# Implementation

# Implementation

- Easy to implement multistage proving, since we do not alter the linear logic or proof algorithm.
- We use a script to pass premises to the Glue Semantics Workbench (GSW) (Messmer & Zymla 2018) in a recursive fashion.
- GSW can only prove non-atomic-typed goals if the goal type is known in advance.
- This can be an advantage when dealing with messy, real-world data.
  - If we know the goal type, we can then provide a dummy meaning side if that sub-proof fails, so avoiding *global* proof failure.

# Messy data

(26) Kim told him [the er, y'know ... what did- was it the security code or something?]

(27)

$$\begin{array}{|l|}\hline e\ x\ y \\ \hline \mathit{tell}(e) \\ \mathit{Agent}(e, x) \\ \mathit{Named}(x, \mathrm{kim}) \\ \mathit{Goal}(e, y) \\ \mathit{male}(y) \\ y = ? \\ \mathit{Proposition}(e, \mathrm{DUMMY\_PROP}) \\ \hline \end{array}$$

# Setting a goal

- Setting an explicit goal for each sub-proof also ensures the proving process remains parallel rather than sequential.
  - (That is, we don't have to prove a sub-tree in order for its mother to use it in another proof, since its type is given from the start.)

(28)
$$\mathrm{N} \rightarrow \quad \begin{array}{c} \widehat{\mathrm{Adj}} \\ \downarrow \in (\uparrow \textsc{adj}) \\ \hat{*}_\gamma = *_\gamma \end{array} \qquad \begin{array}{c} \mathrm{N} \\ \uparrow = \downarrow \\ \hat{*}_\gamma \lhd *_\gamma \\ \boxed{\pi_2(*_\gamma) = e_\uparrow \multimap t_\uparrow} \end{array}$$

(29)
$$\begin{tikzcd}[column sep=small]
 & \bullet = _ : e_c \multimap t_c \arrow[dl, dash] \arrow[dr, dash] & & \\
\begin{array}{c}\lambda P \lambda x.\mathbf{trustworthy}(x) \wedge P(x) : \\ (e_c \multimap t_c) \multimap e_c \multimap t_c\end{array} & & \bullet = _ : e_c \multimap t_c \arrow[dl, dash] \arrow[dr, dash] & \\
 & \begin{array}{c}\lambda P \lambda x.\mathbf{former}(P(x)) : \\ (e_c \multimap t_c) \multimap e_c \multimap t_c\end{array} & & \bullet = _ : e_c \multimap t_c \arrow[d, dash] \\
 & & & \begin{array}{c}\lambda x.\mathbf{chairman}(x) : \\ e_c \multimap t_c\end{array}
\end{tikzcd}$$

# Conclusion

# Conclusion

- The permissive approach to scope interactions is a characteristic property of Glue.
  - Sets it apart from e.g. syntactic approaches which use Quantifier Raising.
- But it has been recognised since the start that this freedom needs to be constrained.
- Unlike several previous approaches, our proposal makes use of the modular projection architecture rather than modifications to the linear logic side.
- This allows us to control the structure of linear logic proofs, and determine which parts of a proof should be (in)accessible to others.
  - This gives both theoretical and practical advantages.

# References I


Andrews, Avery D. 2018. Sets, heads and spreading in LFG. *Journal of Language Modelling* 6(1). 131–174. doi:10.15398/jlm.v6i1.175.

Andrews, Avery D. & Christopher D. Manning. 1999. *Complex predicates and information spreading in LFG*. Stanford, CA: CSLI Publications.

Asudeh, Ash, Gianluca Giorgolo & Ida Toivonen. 2014. Meaning and valency. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the LFG14 Conference*, 68–88. CSLI Publications. http://web.stanford.edu/group/cslipublications/cslipublications/LFG/19/papers/lfg14asudehetal.pdf.

Barker, Chris. 2021. Rethinking scope islands. *Linguistic Inquiry* 1–29.

Campbell, Richard. 2002. Computation of modifier scope in NP by a language-neutral method. In *COLING 2002: the 19th international conference on computational linguistics*, https://aclanthology.org/C02-1043.

Crouch, Richard & Josef van Genabith. 1999. Context change, underspecification and the structure of Glue language derivations. In Mary Dalrymple (ed.), *Semantics and syntax in Lexical Functional Grammar*, 117–189. Cambridge, MA: MIT Press.

Findlay, Jamie Y. 2020. Mapping Theory and the anatomy of a lexical entry. In Miriam Butt & Ida Toivonen (eds.), *Proceedings of the LFG20 Conference*, 127–147. Stanford, CA: CSLI Publications. http://web.stanford.edu/group/cslipublications/cslipublications/LFG/LFG-2020/lfg2020-findlay.pdf.

Gotham, Matthew. 2019. Constraining scope ambiguity in LFG+Glue. In Miriam Butt, Tracy Holloway King & Ida Toivonen (eds.), *Proceedings of the LFG'19 Conference*, Stanford, CA: CSLI Publications. http://csli-publications.stanford.edu/LFG/2019.

Gotham, Matthew. 2021. Approaches to scope islands in LFG+Glue. In Miriam Butt, Jamie Y. Findlay & Ida Toivonen (eds.), *Proceedings of the LFG'21 Conference*, Stanford, CA: CSLI Publications.

# References II


Hepple, Mark. 1996. A compilation-chart method for linear categorial deduction. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, .

Lev, Iddo. 2007. *Packed computation of exact meaning representations*: Stanford University dissertation.

Lowe, John J. 2015. Complex predicates: an LFG + glue analysis. *Journal of Language Modelling* 3(2). 413–462. http://jlm.ipipan.waw.pl/index.php/JLM/article/view/125/116.

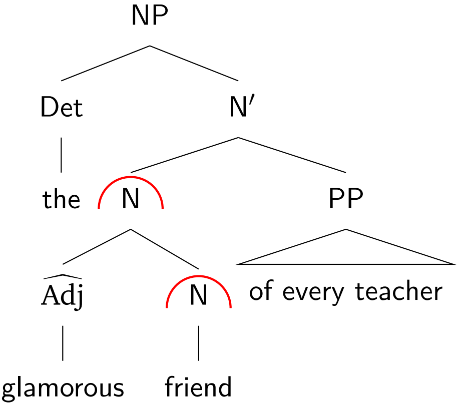
Messmer, Moritz & Mark-Matthias Zymla. 2018. The glue semantics workbench: A modular toolkit for exploring linear logic and glue semantics. In Miriam Butt & Tracy Holloway King (eds.), *Proceedings of the lfg'18 conference, university of vienna*, 249–263. Stanford, CA: CSLI Publications. http://cslipublications.stanford.edu/LFG/2018/lfg2018-messmer-zymla.pdf.

Przepiórkowski, Adam. 2017. A full-fledged hierarchical lexicon in LFG: the FrameNet approach. In Victoria Rosén & Koenraad De Smedt (eds.), *The very model of a modern linguist: in honor of Helge Dyvik* (Bergen Language and Linguistics Studies 8), 202–219. Bergen: University of Bergen. https://doi.org/10.15845/bells.v8i1.1336.

# Quantifiers inside NPs

- Does our adjective rule make NPs scope islands?
- No, because we are assuming head-adjunction of the adjective – only the N itself is an island, and complement PPs will attach higher:

(30)



# Scope islands

- Still unclear exactly what determines when a quantifier can escape from a particular island. (Barker 2021)
  - **Scope Island Subset Constraint:** given any two scope islands, the scope-takers trapped by one is a subset of the scope-takers trapped by the other. (Barker 2021)
- Quantifier:

(31) $(\uparrow_\sigma \textsc{EscaperStrength}) = m$

- Embedding verb:

(32) $((\uparrow \textsc{comp})_\sigma \textsc{IslandStrength}) = n$

(33) $((\uparrow \textsc{comp})_\sigma \textsc{IslandStrength}) > ((\uparrow \textsc{comp})_\sigma \textsc{EscaperStrength}) \Rightarrow ((\uparrow \textsc{comp})_\sigma \textsc{ScopeIsland}) = +$

# Ambiguities

$$\bullet = \left\{ \begin{array}{l} \textbf{a}(z, \textbf{warden}(z), \textbf{think}(z, \textbf{every}(x, \textbf{prisoner}(x), \textbf{a}(y, \textbf{bird}(y), \textbf{see}(x, y)))))) : t_t, \\ \textbf{a}(z, \textbf{warden}(z), \textbf{think}(z, \textbf{a}(y, \textbf{bird}(y), \textbf{every}(x, \textbf{prisoner}(x), \textbf{see}(x, y)))))) : t_t \end{array} \right\}$$

$$\begin{array}{c} \lambda R.\lambda S.\textbf{a}(z, R(z), S(z)) : \\ (e_w \multimap t_w) \multimap \forall H[(e_w \multimap t_H) \multimap t_H] \end{array} \quad \begin{array}{c} \lambda x.\textbf{warden}(x) : \\ e_w \multimap t_w \end{array} \quad \begin{array}{c} \lambda p.\lambda x.\textbf{think}(x, p) : \\ t_s \multimap e_w \multimap t_t \end{array} \quad \bullet = \left\{ \begin{array}{l} \textbf{every}(x, \textbf{prisoner}(x), \textbf{a}(y, \textbf{bird}(y), \textbf{see}(x, y))) : t_s, \\ \textbf{a}(y, \textbf{bird}(y), \textbf{every}(x, \textbf{prisoner}(x), \textbf{see}(x, y))) : t_s \end{array} \right\}$$

$$\begin{array}{c} \lambda R.\lambda S.\textbf{every}(x, R(x), S(x)) : \\ (e_p \multimap t_p) \multimap \forall H[(e_p \multimap t_H) \multimap t_H] \end{array} \quad \begin{array}{c} \lambda x.\textbf{prisoner}(x) : \\ e_p \multimap t_p \end{array} \quad \begin{array}{c} \lambda y.\lambda x.\textbf{see}(x, y) : \\ e_b \multimap e_p \multimap t_e \end{array} \quad \begin{array}{c} \lambda R.\lambda S.\textbf{a}(y, R(y), S(y)) : \\ (e_b \multimap t_b) \multimap \forall H[(e_b \multimap t_H) \multimap t_H] \end{array} \quad \begin{array}{c} \lambda y.\textbf{bird}(x) : \\ e_b \multimap t_b \end{array}$$

# Scope freezing

- English double object construction/German SVO clauses: surface scope only (Gotham 2019: 115).

(34) Hilary gave a student every grade.

(35) Ein Polizist bewacht jeden Ausgang.
*a*.NOM *police.officer guards every*.ACC *exit*
'A police officer guards every exit.'

- Add articulation to the relevant PSRs:

(36)
$$\text{VP} \rightarrow \quad \begin{array}{c} \text{V} \\ \uparrow = \downarrow \\ \hat{*}_\gamma \lhd *_\gamma \\ *_\gamma = \%v \end{array} \quad \begin{array}{c} \text{NP} \\ (\uparrow \text{OBJ}) = \downarrow \\ \hat{*}_\gamma = *_\gamma \end{array} \quad \begin{array}{c} \text{NP} \\ (\uparrow \text{OBJ}_\theta) = \downarrow \\ \hat{*}_\gamma \lhd *_\gamma \\ *_\gamma = \%v \end{array}$$

## Sublexical meanings

- This approach could be applied to many but not all sublexical meanings.
- Adjective meaning needs to be available for adverbial modification, so cannot be isolated in a sub-proof.

(37) trustworthy $\widehat{\text{Adj}}$

$(\uparrow \text{PRED}) = \text{`trustworthy'}$

$\%N = (\text{ADJ} \in \uparrow)$

$\hat{*}_\gamma \lhd \lambda x.\mathbf{trustworthy}(x) : e_\uparrow \multimap t_\uparrow$

$\hat{*}_\gamma \lhd \lambda Q \lambda P \lambda x.P(x) \wedge Q(x) :$

$(e_\uparrow \multimap t_\uparrow) \multimap (e_{\%N} \multimap t_{\%N}) \multimap (e_{\%N} \multimap t_{\%N})$

(38) very Adv

$(\uparrow \text{PRED}) = \text{`very'}$

$\%A = (\text{ADJ} \in \uparrow)$

$\hat{*}_\gamma \lhd \lambda P.\mathbf{very}(P) : (e_{\%A} \multimap t_{\%A}) \multimap (e_{\%A} \multimap t_{\%A})$