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

32nd South of England LFG Meeting

21 May, 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**))

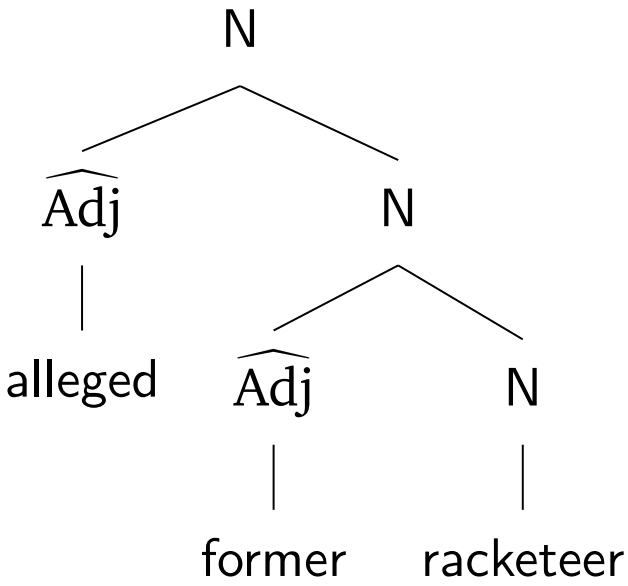
(2) a counterfeit American coin (Campbell 2002)
$\Rightarrow$ **counterfeit**(**american**(**coin**))
$\not\Rightarrow$ **american**(**counterfeit**(**coin**))

(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.



$$r\begin{bmatrix}\text{PRED} & \text{'racketeer'} \\ \text{ADJ} & \left\{\begin{bmatrix}\text{PRED} & \text{'alleged'}\end{bmatrix}\right. \\ & \left.\begin{bmatrix}\text{PRED} & \text{'former'}\end{bmatrix}\right\}\end{bmatrix}$$

racketeer : $(e_r \multimap t_r)$
alleged : $(e_r \multimap t_r) \multimap (e_r \multimap t_r)$
former : $(e_r \multimap t_r) \multimap (e_r \multimap t_r)$

$\Rightarrow$ **former**(**alleged**(**racketeer**))
alleged(**former**(**racketeer**))

Three scope problems

2. Scope islands:

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

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.

$(^{*}\forall > \exists)$

(6) **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.

$(\forall > \exists)$

Scope islands

- Not related to 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 $(\uparrow \textsc{pred}) = \text{`crush'}$
$@\textsc{Agent}$
$@\textsc{Patient}$
$@\textsc{Passive}$
$\lambda e.\mathbf{crush}(e) : v_{\uparrow} \multimap t_{\uparrow}$

(after Asudeh et al. 2014)

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

Sublexical meanings

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

(9) $\textsc{Agent}{:=}$
$\lambda P.\lambda x.\lambda e.P(e) \wedge \textbf{agent}(e,x) : (v_{\uparrow} \multimap t_{\uparrow}) \multimap e_{(\uparrow \textsc{subj})} \multimap v_{\uparrow} \multimap t_{\uparrow}$[2]

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

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

[2] We simplify by resolving the mapping of ARG1 and ARG2 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*.

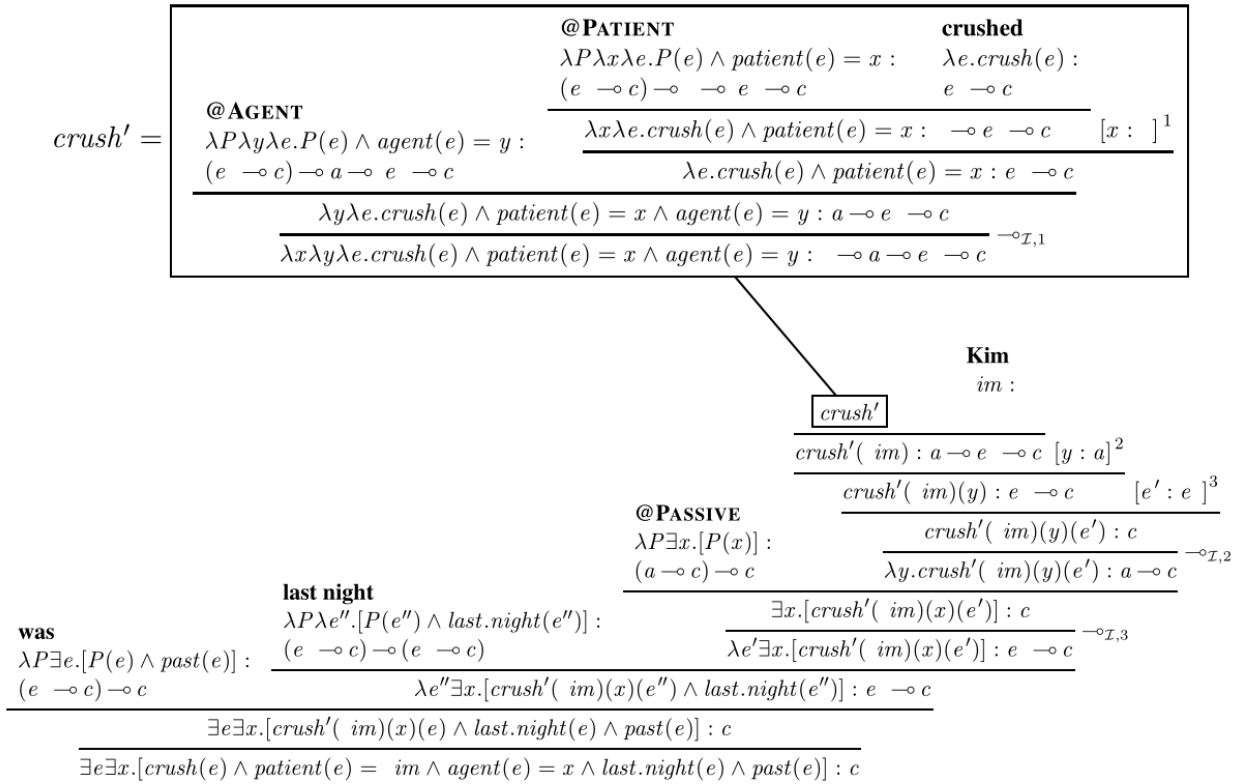


Figure 7: Proof for *Kim was crushed last night.*

- 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 within LFG 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:

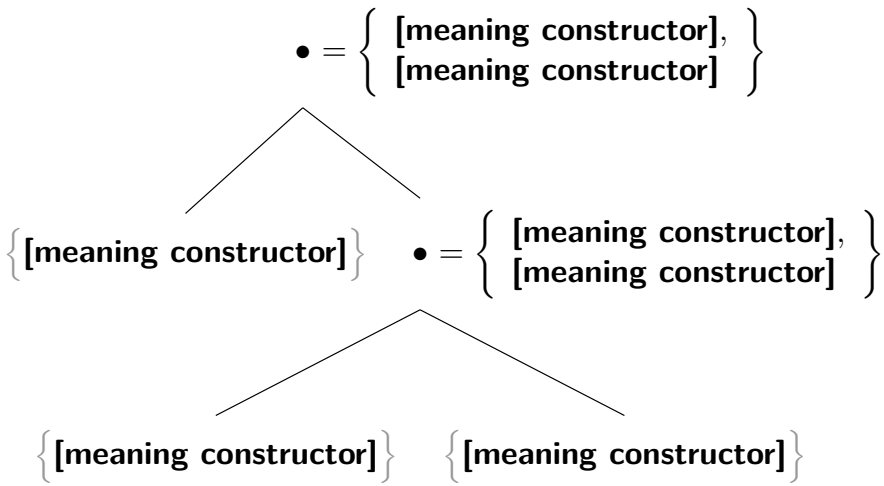
- A tree, where each sub-tree represents a sub-proof.
- Nodes are (sets of) premises (i.e. meaning constructors).
- One meaning constructor from each daughter node is used in the proof of their mother.
- Projected from c-structure via a function Γ.
- Lexically-contributed meaning constructors are introduced 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)
$$\text{S} \rightarrow \underset{\substack{(\uparrow \text{SUBJ}) = \downarrow \\ \hat{*}_\Gamma = *_\Gamma}}{\text{NP}} \quad \underset{\substack{\uparrow = \downarrow \\ \hat{*}_\Gamma = *_\Gamma}}{\text{VP}}$$

- But this can be selectively interrupted!



Modifier scope

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

(14) $$\mathrm{N} \rightarrow \begin{array}{c} \widehat{\mathrm{Adj}} \\ \downarrow \in (\uparrow \mathrm{ADJ}) \\ \hat{*}_\Gamma = *_\Gamma \end{array} \qquad \begin{array}{c} \mathrm{N} \\ \uparrow = \downarrow \\ \hat{*}_\Gamma \lhd *_\Gamma \end{array}$$

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

Modifier scope: lexical entries

(15) former $\widehat{\text{Adj}}$

$(\uparrow \text{PRED}) = \text{‘former’}$

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

$\hat{*}_{\Gamma} \lhd \lambda P \lambda x.\textbf{former}(P(x)) : (e_{\%N} \multimap t_{\%N}) \multimap e_{\%N} \multimap t_{\%N}$

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

$(\uparrow \text{PRED}) = \text{‘trustworthy’}$

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

$\hat{*}_{\Gamma} \lhd \lambda P \lambda x.\textbf{trustworthy}(x) \wedge P(x) : (e_{\%N} \multimap t_{\%N}) \multimap e_{\%N} \multimap t_{\%N}$

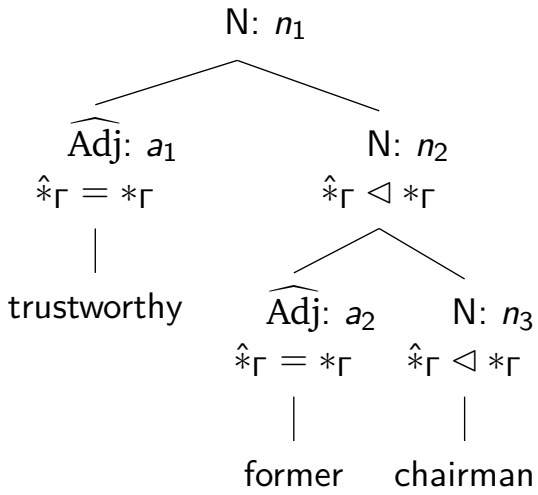
(17) chairman N

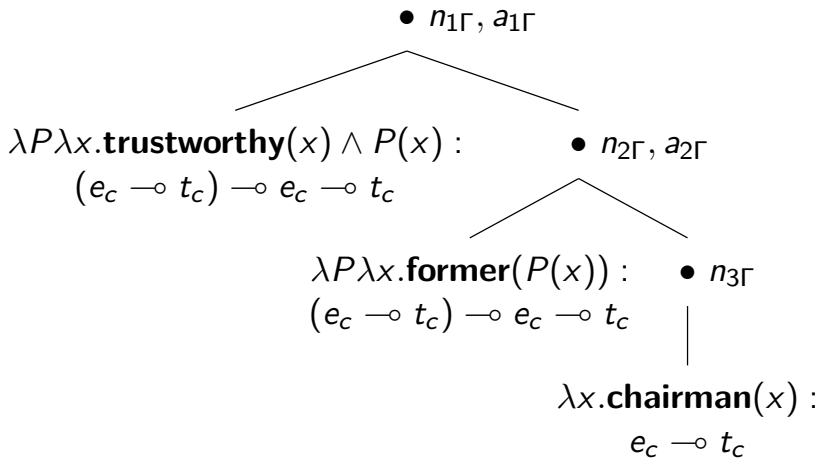
$(\uparrow \text{PRED}) = \text{‘chairman’}$

$\hat{*}_{\Gamma} \lhd \lambda x.\textbf{chairman}(x) : e_{\uparrow} \multimap t_{\uparrow}$

Modifier scope: structures

(18)



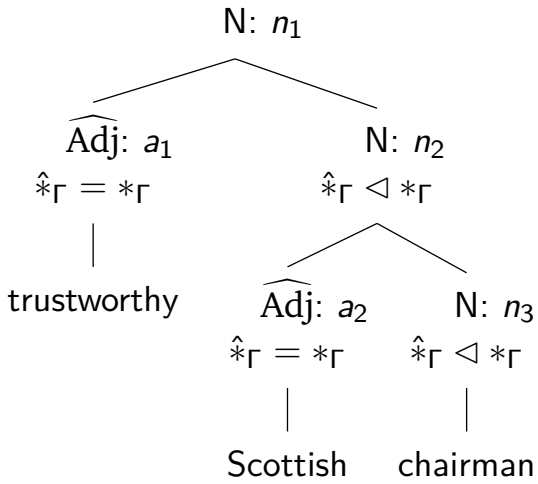


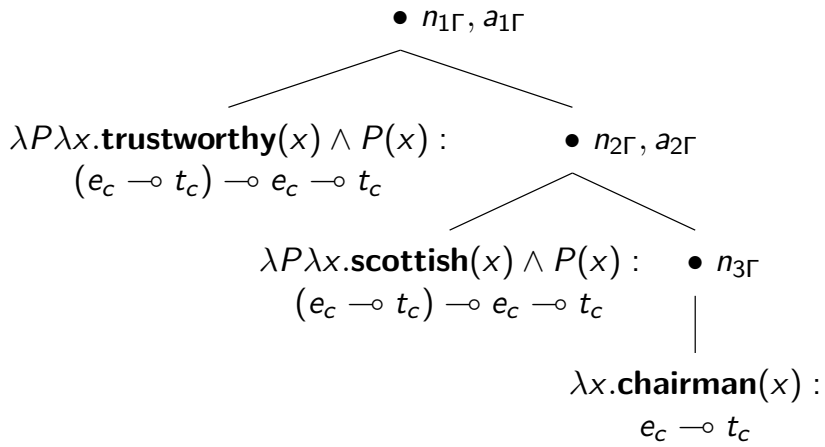
$\Rightarrow$ **trustworthy(former(chairman))**

$\not\Rightarrow$ **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. $\neq$ For every prisoner, there is a warden who thinks they escaped.
$(^{*}\forall > \exists)$

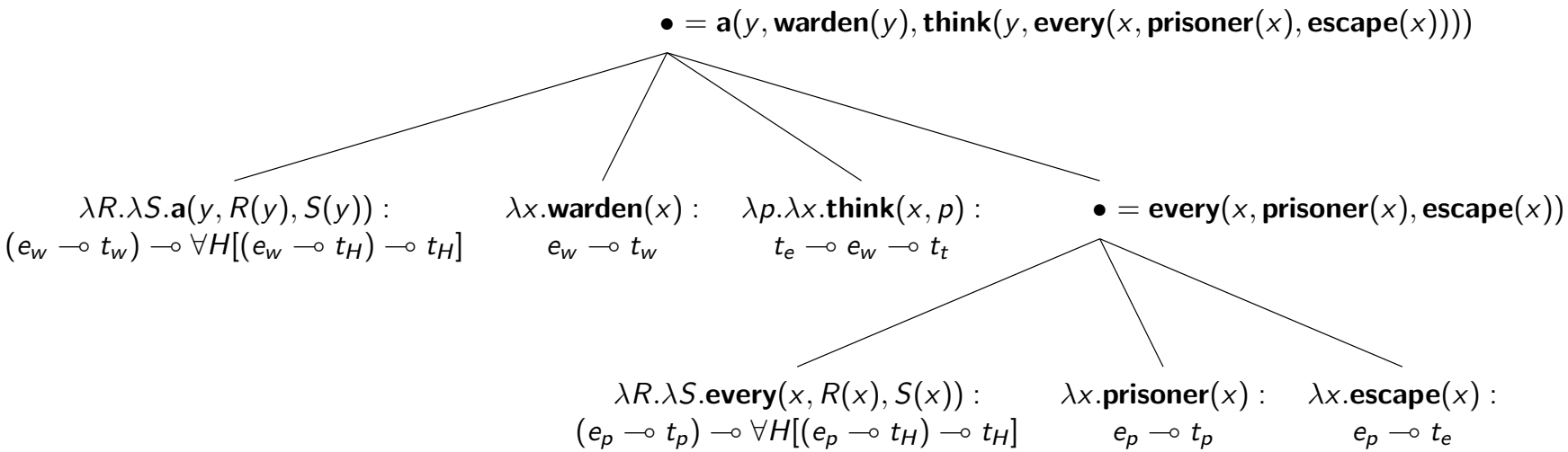
Scope islands

- 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)

$V' \rightarrow$	V	CP
	$\uparrow = \downarrow$	$(\uparrow \textsc{comp}) = \downarrow$
	$\hat{*}_\Gamma = *_\Gamma$	$\hat{*}_\Gamma \lhd *_\Gamma$

Scope islands



Scope islands

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

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

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

$(\uparrow \textsc{pred}) = \text{`crush'}$

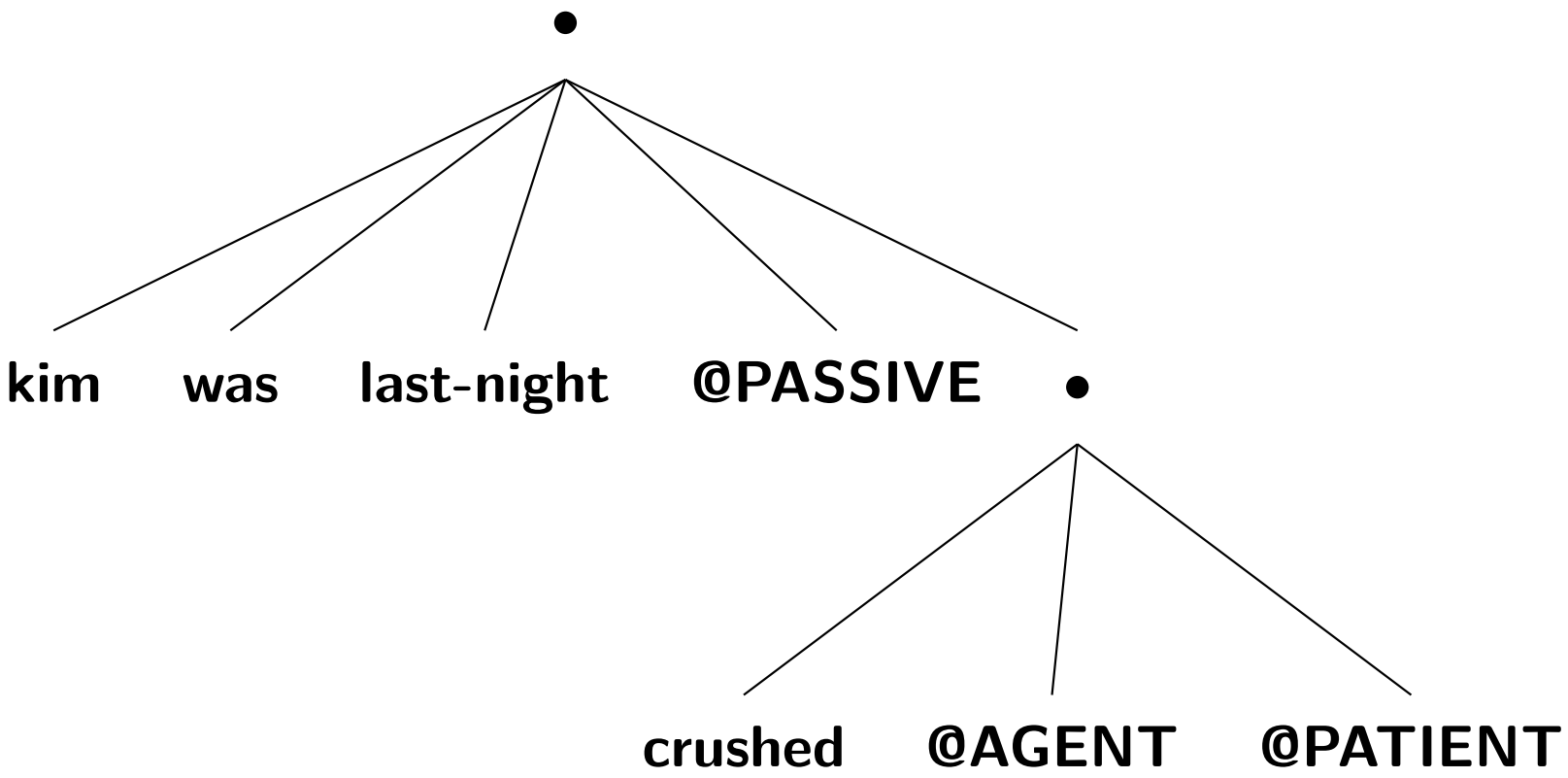
$\hat{*}_\Gamma \lhd *_\Gamma$

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

@Passive & @Agent & @Patient

Sublexical meanings

(25)



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

Sublexical meanings

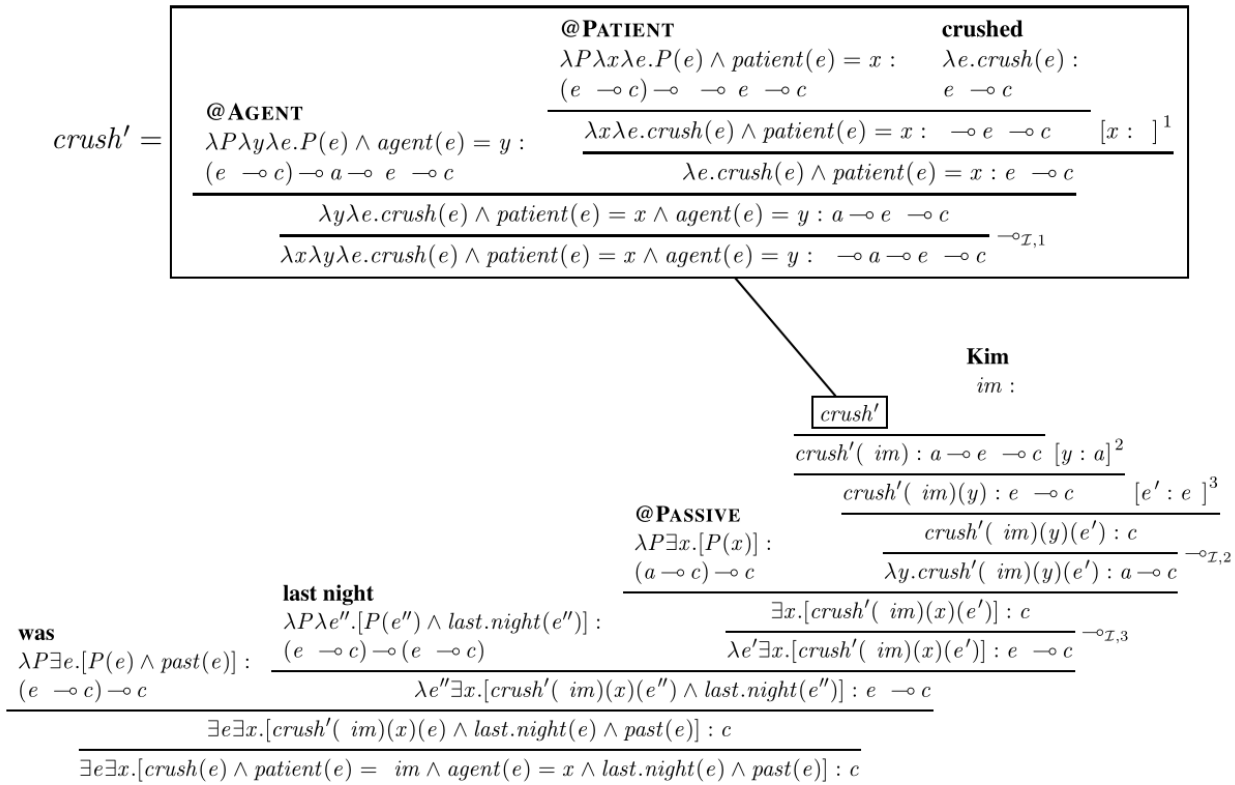


Figure 7: Proof for *Kim was crushed last night.*

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.
- But 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.

Implementation: messy data

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

(27)

e x y
$tell(e)$
$Agent(e, x)$
$Named(x, \mathrm{kim})$
$Goal(e, y)$
$male(y)$
$y = ?$
$Proposition(e, \mathrm{DUMMY\_PROP})$

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.

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:

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

- Embedding verb:

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

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

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.

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

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

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

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

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

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

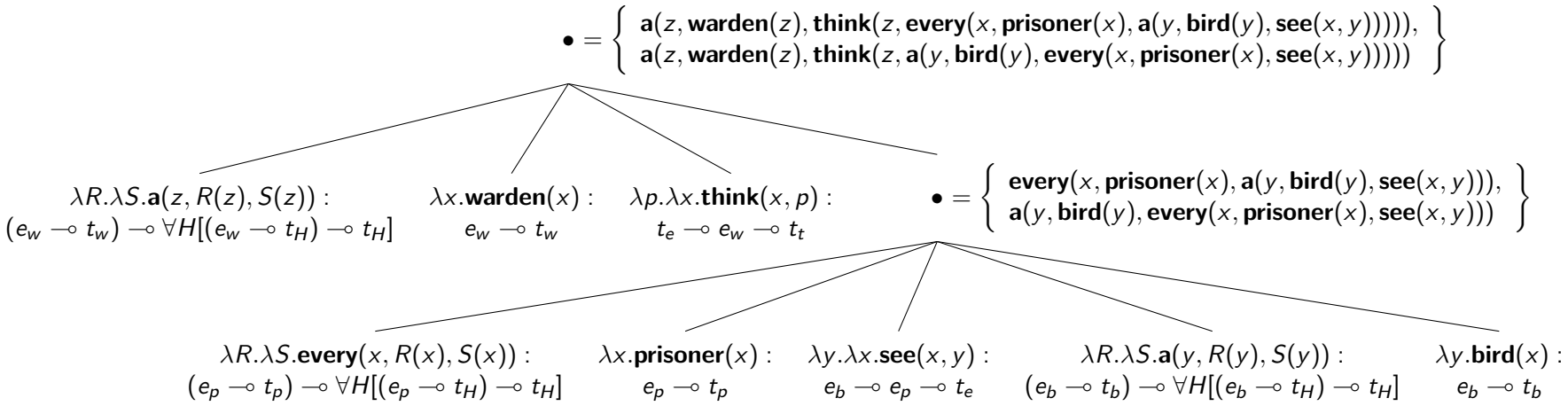
(32) very Adv

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

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

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

Ambiguities



Scope freezing

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

(33) Hilary gave a student every grade.

(34) 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:

(35) $\text{VP} \rightarrow$ V: $\uparrow{=}\downarrow$, $\hat{*}_\Gamma \lhd *_\Gamma$, $*_\Gamma = \%v$ — NP: $(\uparrow \text{OBJ}) = \downarrow$, $\hat{*}_\Gamma = *_\Gamma$ — NP: $(\uparrow \text{OBJ}_\theta) = \downarrow$, $\hat{*}_\Gamma \lhd *_\Gamma$, $*_\Gamma = \%v$