Compositional Neural Semantic Graph Parsing

TaCoS 29

Matthias Lindemann

Based on joint work with
Jonas Groschwitz, Alexander Koller,
Meaghan Fowlie and Mark Johnson

June 21, 2019

What do we want to do

We want to obtain **symbolic** meaning representations of sentences.

- interpretable for experts (that is: you!)
- useful for NLP applications like dialog system, question answering, etc.

In this talk:

- we focus on meaning representation in form of **semantic graphs**.
- we approach **semantic parsing**: (English) sentence $\rightarrow$ meaning representation
- we do this for several kinds of semantic graphs.

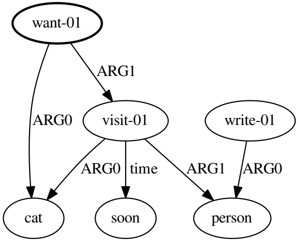
Outline

Semantic graphs

Abstract Meaning Representation (Banarescu et al., 2013)

Abstract Meaning Representation (AMR) is a sentence-level **broad coverage** meaning representation language.
AMRs

- are directed, acyclic and labeled **graphs**
- capture semantic relations, e.g. predicate argument structure
- abstract away from syntactic surface

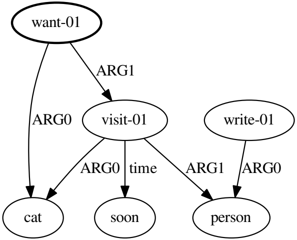


The cat wants to visit the author soon

Abstract Meaning Representation (Banarescu et al., 2013)

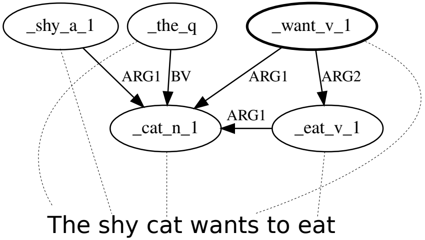
Abstract Meaning Representation (AMR) is a sentence-level **broad coverage** meaning representation language.
AMRs

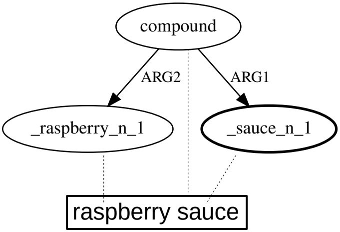
- are directed, acyclic and labeled **graphs**
- capture semantic relations, e.g. predicate argument structure
- abstract away from syntactic surface



The cat wants to visit the author soon
The cats wanted to visit the writer soon

Elementary Dependency Structures (Oepen and Lønning, 2006)

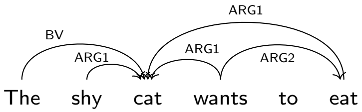




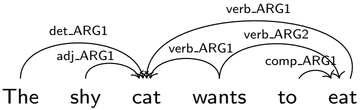
- derived from semantic annotations produced by English Resource Grammar (Flickinger, 2000), a broad-coverage HPSG grammar
- nodes are aligned to words or phrases.

Semantic Dependencies (Oepen et al., 2015)

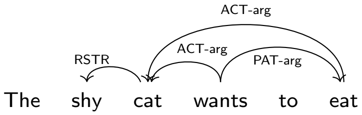
- DM, derived from EDS



- PAS



- PSD



- automatically derived from annotations on WSJ
- nodes are the words from the sentence

Comparison of kinds of semantic graphs

Name	Alignments	Nodes $\approx$ Words	Similarity to surface
AMR	✗	✗	potentially low
EDS	1-many	✗	high
DM	1-1	✓	high
PAS	1-1	✓	high
PSD	1-1	✓	high

- handle certain phenomena differently, e.g. copula and coordination

Deriving semantic graphs

Compositionally deriving graphs

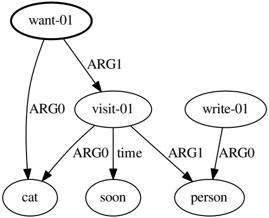
Principle of **Compositionality**

The meaning of a **complex expression** is determined by **meanings of its sub-expressions** and the **rules used to combine** them.

Compositionally deriving graphs

Principle of **Compositionality**

The meaning of a **complex expression** is determined by **meanings of its sub-expressions** and the **rules used to combine** them.



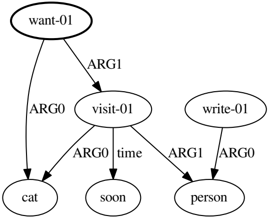
The cat wants to visit the author soon

We know that meaning is constructed compositionally.
. . . but we only see the result of the compositional process.

Compositionally deriving graphs

Principle of Compositionality

The meaning of a **complex expression** is determined by **meanings of its sub-expressions** and the **rules used to combine** them.



The cat wants to visit the author soon

We know that meaning is constructed compositionally.
. . . but we only see the result of the compositional process.
So, we must come up with such a process!

Formal perspective

What kind of formalism do we need to **compositionally** derive **graphs**?

Formal perspective

What kind of formalism do we need to **compositionally** derive **graphs**?

Formalism	**Derivation structure**	**Derived structure**
CFG	Tree	String
TSG,TAG	Tree	Tree
?	Tree	Graph

Formal perspective

What kind of formalism do we need to **compositionally** derive **graphs**?

Formalism	**Derivation structure**	**Derived structure**
CFG	Tree	String
TSG,TAG	Tree	Tree
Graph Algebra	Tree	Graph

Formal perspective

What kind of formalism do we need to **compositionally** derive **graphs**?

Formalism	**Derivation structure**	**Derived structure**
CFG	Tree	String
TSG,TAG	Tree	Tree
Graph Algebra	Tree	Graph

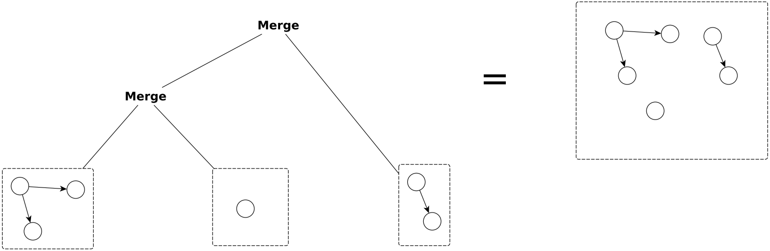
Algebra	**Domain**
Elementary Algebra	$\mathbb{R}$
Boolean Algebra	$\{0, 1\}$
Graph Algebra	Graphs

Formal perspective

What kind of formalism do we need to **compositionally** derive **graphs**?

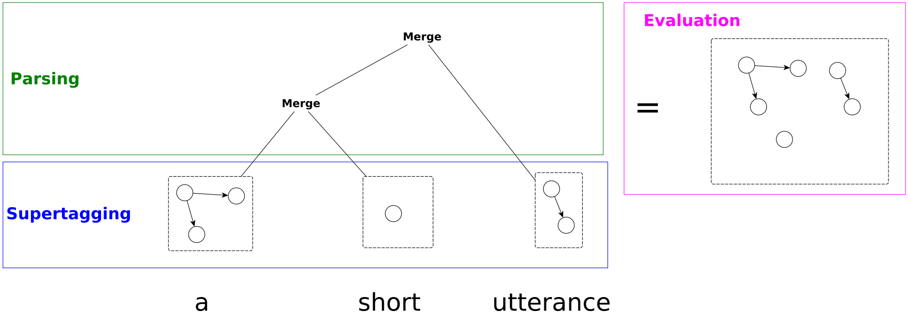
Formalism	Derivation structure	Derived structure
CFG	Tree	String
TSG, TAG	Tree	Tree
Graph Algebra	Tree	Graph

Algebra	Domain
Elementary Algebra	$\mathbb{R}$
Boolean Algebra	$\{0, 1\}$
Graph Algebra	Graphs



$$[\![\mathrm{F}(t_1, t_2, \ldots, t_n)]\!] = [\![\mathrm{F}]\!]([\![t_1]\!], [\![t_2]\!], \ldots, [\![t_n]\!])$$

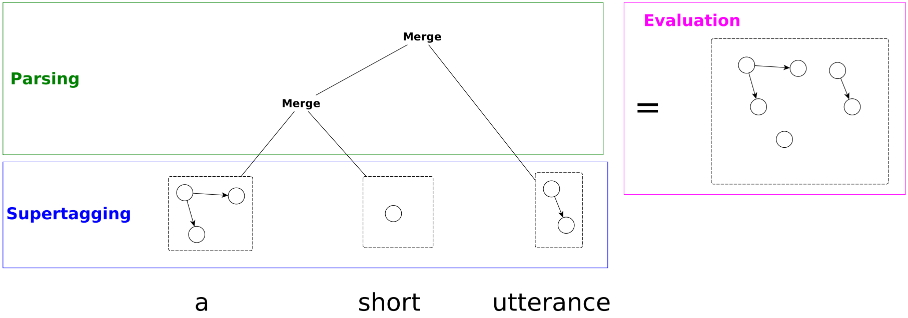
Graph Algebra and Graph Parsing



With a Graph Algebra, we can parse a sentence into a graph by:

- Supertagging: finding an appropriate graph constant for every word
- Parsing: finding a term over all constants
- Evaluating the term

Graph Algebra and Graph Parsing



With a Graph Algebra, we can parse a sentence into a graph by:

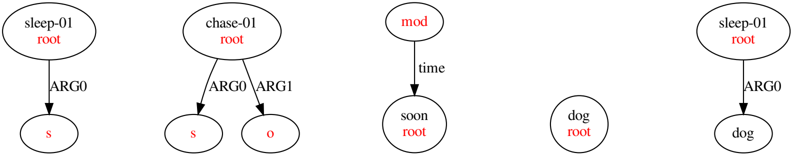
- Supertagging: finding an appropriate graph constant for every word
- Parsing: finding a term over all constants
- Evaluating the term

We need a graph algebra that exploits the compositional structure of our graphs.

AM Algebra (Groschwitz et al., 2017; Groschwitz, 2019)

- linguistically inspired and constrained graph algebra
- built for AMR (but works on the other semantic graphs as well!)
- handles phenomena like control, coordination and relative clauses

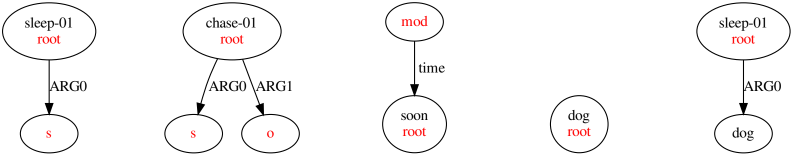
In the AM Algebra, we generalize graphs to *s-graphs*:



AM Algebra (Groschwitz et al., 2017; Groschwitz, 2019)

- linguistically inspired and constrained graph algebra
- built for AMR (but works on the other semantic graphs as well!)
- handles phenomena like control, coordination and relative clauses

In the AM Algebra, we generalize graphs to *s-graphs*:

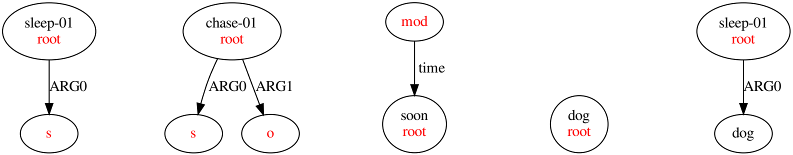


- some nodes of an s-graph (called sources) have an additional name (s for subject, o for object,...)

AM Algebra (Groschwitz et al., 2017; Groschwitz, 2019)

- linguistically inspired and constrained graph algebra
- built for AMR (but works on the other semantic graphs as well!)
- handles phenomena like control, coordination and relative clauses

In the AM Algebra, we generalize graphs to *s-graphs*:

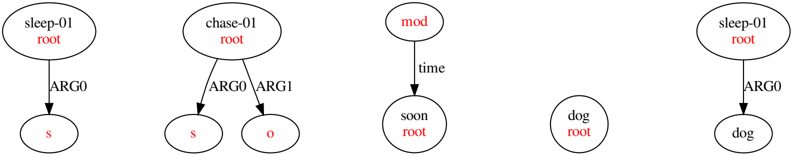


- some nodes of an s-graph (called sources) have an additional name (s for subject, o for object,...)
- all s-graphs have a source called root, which marks the "head"

AM Algebra (Groschwitz et al., 2017; Groschwitz, 2019)

- linguistically inspired and constrained graph algebra
- built for AMR (but works on the other semantic graphs as well!)
- handles phenomena like control, coordination and relative clauses

In the AM Algebra, we generalize graphs to *s-graphs*:

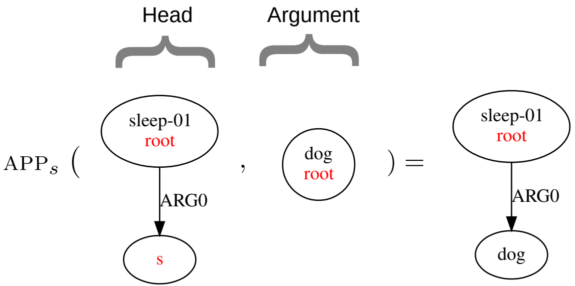


Type [s]　Type [s, o]　Type [mod]　Type []　Type []

- some nodes of an s-graph (called sources) have an additional name (s for subject, o for object,...)
- all s-graphs have a source called root, which marks the "head"
- s-graphs are typed (≈ source names excluding root)

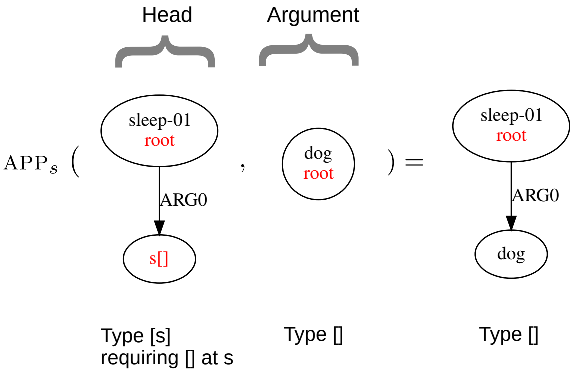
Operations of AM Algebra: Apply

APP_{α} fills argument slot α of the head with root of the argument:



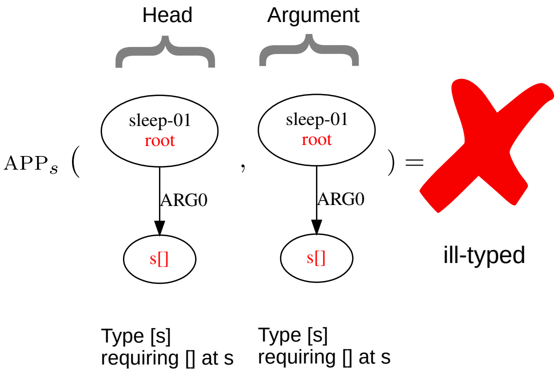
Operations of AM Algebra: Apply

APP_α fills argument slot α of the head with root of the argument:

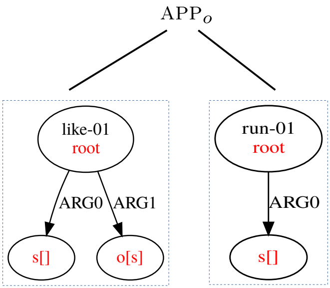


Operations of AM Algebra: Apply

APP_α fills argument slot α of the head with root of the argument:



APP can create reentrancy

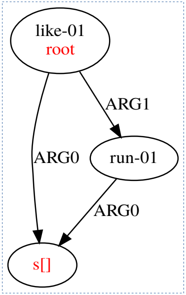


Type [s,o[s]]
requiring [] at s
requiring [s] at o

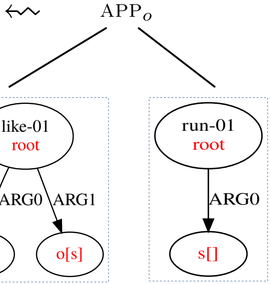
Type [s]
requiring [] at s

The fox likes running.

APP can create reentrancy



Type [s]
requiring [] at s



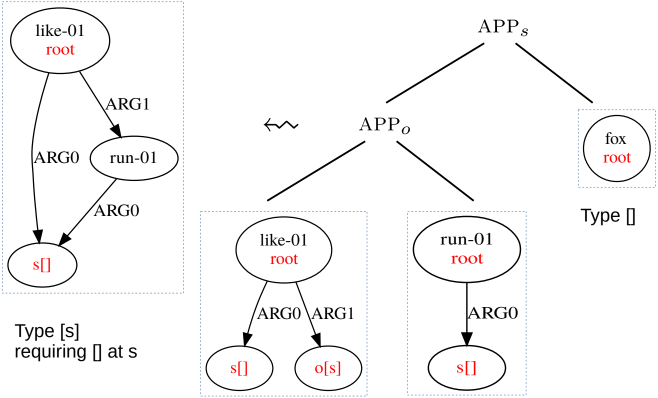
Type [s,o[s]]
requiring [] at s
requiring [s] at o

Type [s]
requiring [] at s

The fox likes running.

Sources with same names collapse.

APP can create reentrancy



Type [s]
requiring [] at s

Type []

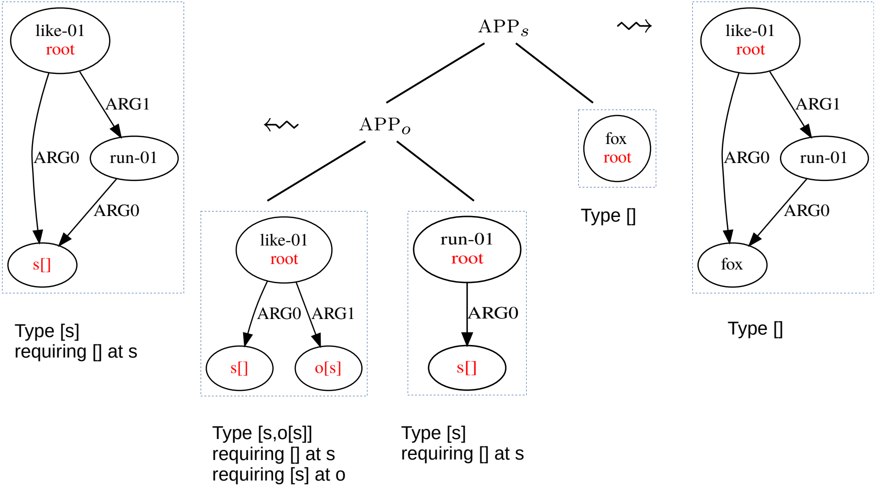
Type [s,o[s]]
requiring [] at s
requiring [s] at o

Type [s]
requiring [] at s

The fox likes running.

Sources with same names collapse.

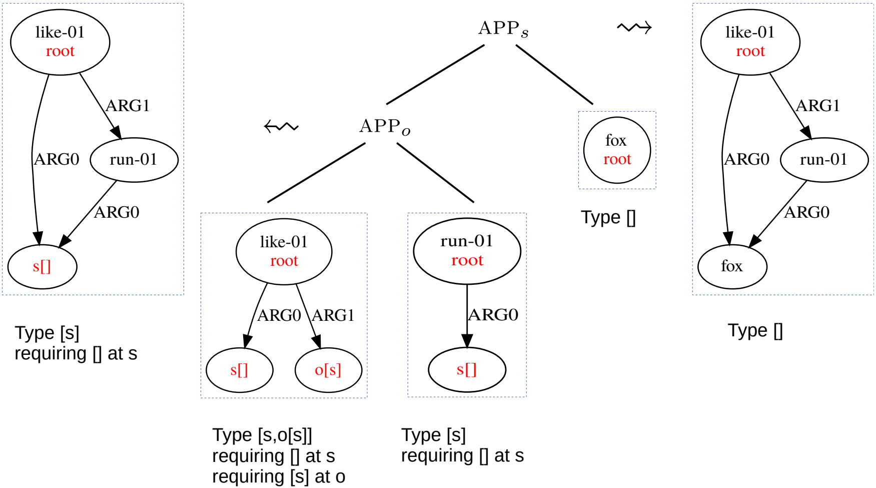
APP can create reentrancy



The fox likes running.

Sources with same names collapse.

APP can create reentrancy

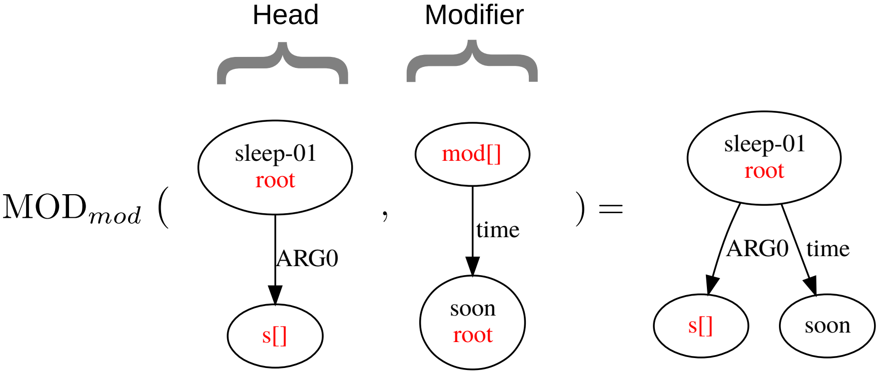


The fox likes running.

Sources with same names collapse.
This constant for *likes* enforces reentrancy!

Operations of AM Algebra: Modify

MOD_{α} plugs α-source of the modifier into the root of the head:



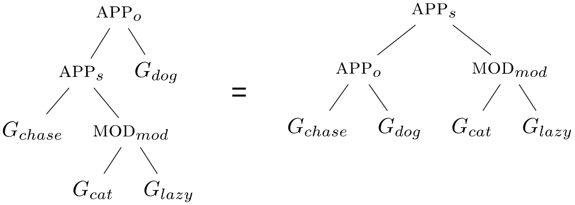
Type [s]
requiring [] at s

Type [mod]
requiring [] at mod

Type [s]
requiring [] at s

Spurious Ambiguity

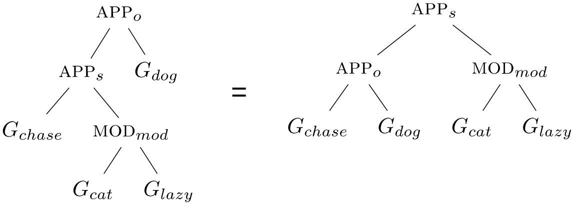
Order of operations does not always matter:



The lazy cat chases the dog

Spurious Ambiguity

Order of operations does not always matter:

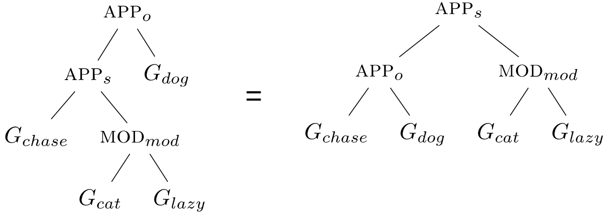


The lazy cat chases the dog

Problem:
We predict terms instead of graphs. Which term shall we pick for training the parser?

AM Dependency Trees

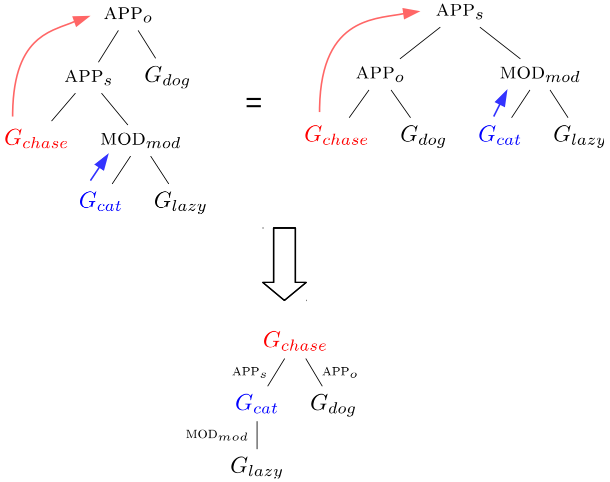
Solution: TRANSFORM terms to dependency trees with *unspecified order*.



The lazy cat chases the dog

AM Dependency Trees

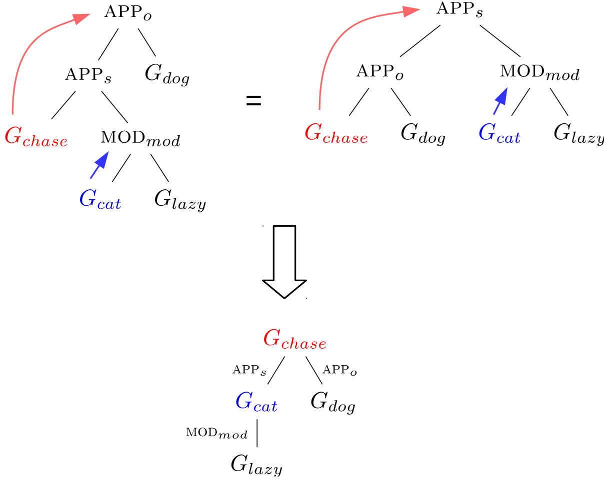
Solution: TRANSFORM terms to dependency trees with *unspecified order*.
Works like constituent tree $\rightarrow$ dependency tree in syntax.



The lazy cat chases the dog

AM Dependency Trees

Solution: TRANSFORM terms to dependency trees with *unspecified order*.
Works like constituent tree → dependency tree in syntax.



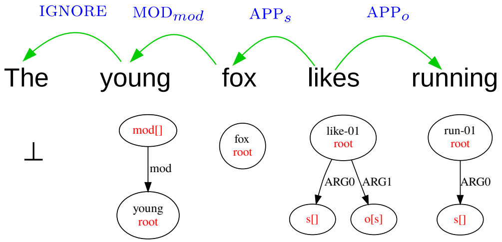
The lazy cat chases the dog

Note: it can be proven that an AM dependency tree *uniquely* denotes a graph (Groschwitz, 2019)

Parsing

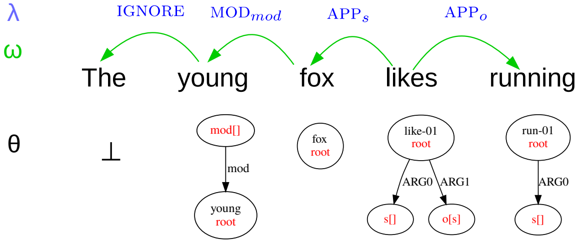
Graph Parsing is now AM Dependency Parsing

Graph Parsing = Supertagging + Dependency Parsing:



Graph Parsing is now AM Dependency Parsing

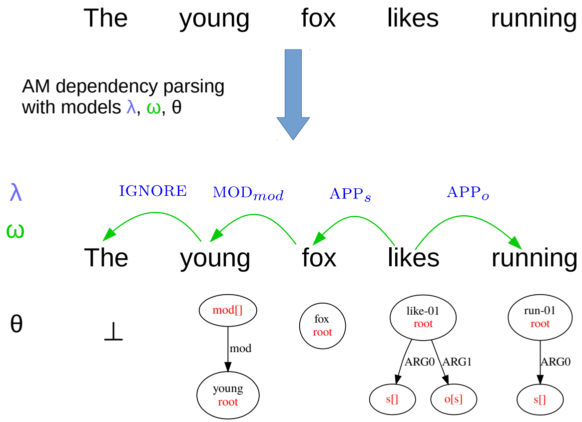
Graph Parsing = Supertagging + Dependency Parsing:



- ω, λ are components of neural dependency parsing model of Kiperwasser and Goldberg (2016)
- θ is a biLSTM supertagger (Groschwitz et al., 2018)

Graph Parsing is now AM Dependency Parsing

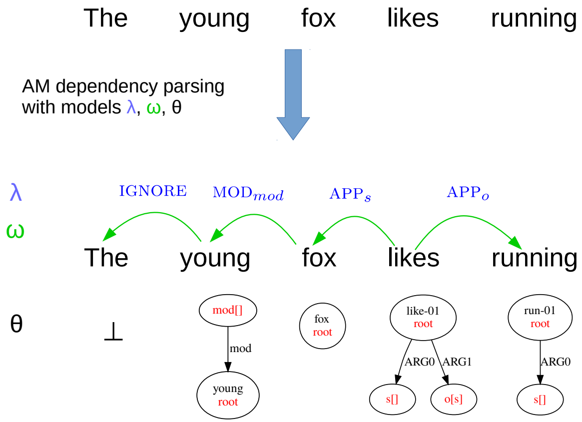
Graph Parsing = Supertagging + Dependency Parsing:



- ω, λ are components of neural dependency parsing model of Kiperwasser and Goldberg (2016)
- θ is a biLSTM supertagger (Groschwitz et al., 2018)
- we want to find the **highest scoring**, **well-typed** AM dependency tree

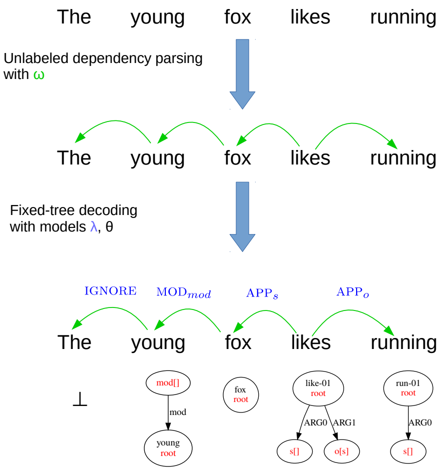
Graph Parsing is now AM Dependency Parsing

Graph Parsing = Supertagging + Dependency Parsing:



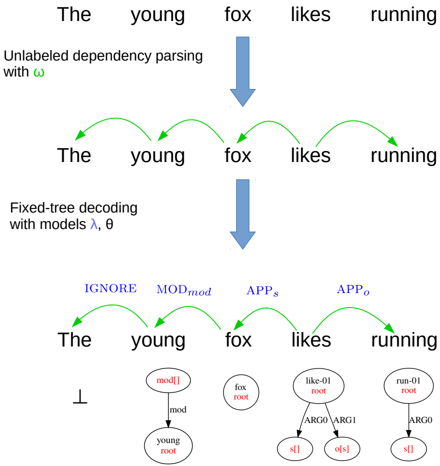
- ω, λ are components of neural dependency parsing model of Kiperwasser and Goldberg (2016)
- θ is a biLSTM supertagger (Groschwitz et al., 2018)
- we want to find the **highest scoring**, **well-typed** AM dependency tree
- **NP-complete!**

Make the problem easier



- (relatively) efficient approach
- fixed-tree decoding guarantees well-typedness

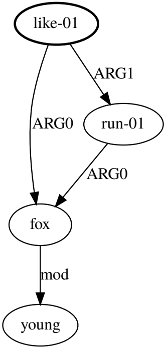
Make the problem easier



- (relatively) efficient approach
- fixed-tree decoding guarantees well-typedness
- if unlabeled tree is bad, no chance to recover – but works well in practice

Corpus creation

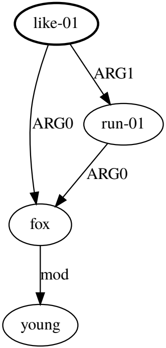
Problem: We only have graphs, but need AM dependency trees and supertags for training



The young fox likes running

Corpus creation

Problem: We only have graphs, but need AM dependency trees and supertags for training

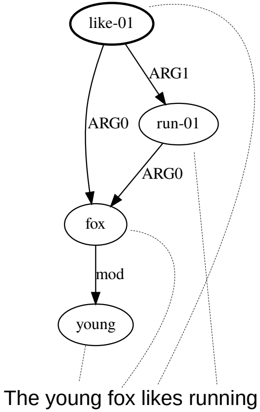


The young fox likes running

Steps:

Corpus creation

Problem: We only have graphs, but need AM dependency trees and supertags for training

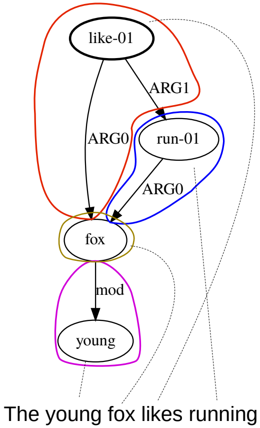


Steps:

- align words to nodes (only required for AMR)

Corpus creation

Problem: We only have graphs, but need AM dependency trees and supertags for training

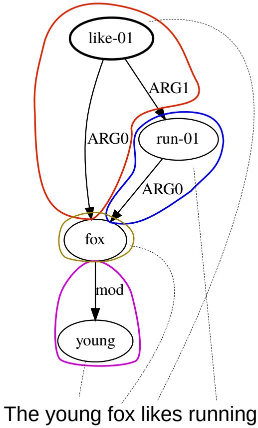


Steps:

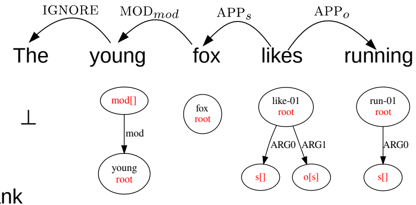
- align words to nodes (only required for AMR)
- identify "blobs": decide where edges belong
- assign sources names and type requirements

Corpus creation

Problem: We only have graphs, but need AM dependency trees and supertags for training



Create treebank



Steps:

- align words to nodes (only required for AMR)
- identify "blobs": decide where edges belong
- assign sources names and type requirements
- find AM dependency tree that evaluates to correct graph

Experiments

Experimental setup

Embeddings:

- Baseline: GloVe (Pennington et al., 2014)
- BERT (Devlin et al., 2019): state-of-the-art *contextualized* word embeddings

Experimental setup

Embeddings:

- Baseline: GloVe (Pennington et al., 2014)
- BERT (Devlin et al., 2019): state-of-the-art *contextualized* word embeddings

Multi-Task Learning:

- Single Task: train on one corpus, evaluate on corresponding test corpus
- Multi-Task: train single model on all corpora + English Universal Dependencies, evaluate on all test corpora

Experimental setup

Embeddings:

- Baseline: GloVe (Pennington et al., 2014)
- BERT (Devlin et al., 2019): state-of-the-art *contextualized* word embeddings

Multi-Task Learning:

- Single Task: train on one corpus, evaluate on corresponding test corpus
- Multi-Task: train single model on all corpora + English Universal Dependencies, evaluate on all test corpora

Evaluation metrics:

- (Smatch) F-score: find best mapping from system output to gold graph, compute overlap
- EDM (only for EDS): also takes alignment into account

Results

	DM		PAS		PSD		EDS		AMR 2015	AMR 2017
	id F	ood F	id F	ood F	id F	ood F	Smatch F	EDM	Smatch F	Smatch F
Lyu and Titov (2018)	-	-	-	-	-	-	-	-	73.7	74.4 ±0.16
Zhang et al. (2019)	-	-	-	-	-	-	-	-	-	**76.3** ±0.1
Peng et al. (2017) Basic	89.4	84.5	92.2	88.3	77.6	75.3	-	-	-	-
Dozat and Manning (2018)[1]	93.7	88.9	94.0	90.8	81.0	79.4	-	-	-	-
Buys and Blunsom (2017)	-	-	-	-	-	-	85.5	85.9	60.1	-
Chen et al. (2018)	-	-	-	-	-	-	**90.9**[2]	**90.4**[3]	-	-
This work (GloVe)	90.4 ±0.2	84.3 ±0.2	91.4 ±0.1	86.6 ±0.1	78.1 ±0.2	74.5 ±0.2	87.6 ±0.1	82.5 ±0.1	69.2 ±0.4	70.7 ±0.2
This work (BERT)	**93.9** ±0.1	**90.3** ±0.1	**94.5** ±0.1	**92.5** ±0.1	**82.0** ±0.1	**81.5** ±0.3	90.1 ±0.1	84.9 ±0.1	**74.3** ±0.2	75.3 ±0.2
Peng et al. (2017) Freda1	90.0	84.9	92.3	88.3	78.1	75.8	-	-	-	-
Peng et al. (2017) Freda3	90.4	85.3	92.7	89.0	78.5	76.4	-	-	-	-
This work, MTL (GloVe)	91.2 ±0.1	85.7 ±0.0	92.2 ±0.2	88.0 ±0.3	78.9 ±0.3	76.2 ±0.4	88.2 ±0.1	83.3 ±0.1	(70.4) ±0.2	71.2 ±0.2
This work, MTL (BERT)	**94.1** ±0.1	**90.5** ±0.1	**94.7** ±0.1	**92.8** ±0.1	**82.1** ±0.2	**81.6** ±0.1	90.4 ±0.1	85.2 ±0.1	(74.5) ±0.1	75.3 ±0.1

[1] Computed macro-averages instead of micro-averages

[2] Uses gold syntax information from the HPSG DeepBank annotations at training time.

Summary

- Semantic graphs can be built and parsed compositionally using AM Algebra

Summary

- Semantic graphs can be built and parsed compositionally using AM Algebra
- Graph parsing can be reformulated as supertagging and constrained dependency parsing

Summary

- Semantic graphs can be built and parsed compositionally using AM Algebra
- Graph parsing can be reformulated as supertagging and constrained dependency parsing
- Experimental results:
  - shows that AM Algebra works well beyond toy examples
  - competitive results
  - considerable gains through MTL
  - BERT (Devlin et al., 2019) extremely boosts performs, set new state-of-the-art on DM, PAS, PSD and AMR 2015.

Summary

- Semantic graphs can be built and parsed compositionally using AM Algebra
- Graph parsing can be reformulated as supertagging and constrained dependency parsing
- Experimental results:
  - shows that AM Algebra works well beyond toy examples
  - competitive results
  - considerable gains through MTL
  - BERT (Devlin et al., 2019) extremely boosts performs, set new state-of-the-art on DM, PAS, PSD and AMR 2015.
- Future research
  - semantic representations like DRT but also SQL for question answering over a database
  - automatic methods for inducing AM dependency trees from graphs

Summary

- Semantic graphs can be built and parsed compositionally using AM Algebra
- Graph parsing can be reformulated as supertagging and constrained dependency parsing
- Experimental results:
  - shows that AM Algebra works well beyond toy examples
  - competitive results
  - considerable gains through MTL
  - BERT (Devlin et al., 2019) extremely boosts performs, set new state-of-the-art on DM, PAS, PSD and AMR 2015.
- Future research
  - semantic representations like DRT but also SQL for question answering over a database
  - automatic methods for inducing AM dependency trees from graphs

Questions?

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*. Association for Computational Linguistics, pages 178–186.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/P17-1112.

Yufei Chen, Weiwei Sun, and Xiaojun Wan. 2018. Accurate SHRG-based semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, pages 408–418. https://www.aclweb.org/anthology/P18-1038.

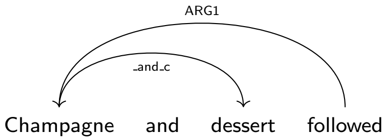
Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. https://www.aclweb.org/anthology/N19-1423.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. http://aclweb.org/anthology/P18-2077.

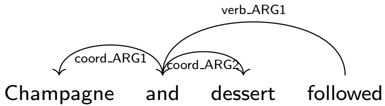
Dan Flickinger. 2000. On building a more effcient grammar by exploiting types. *Natural*

Coordination

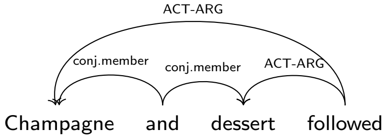
- DM



- PAS



- PSD

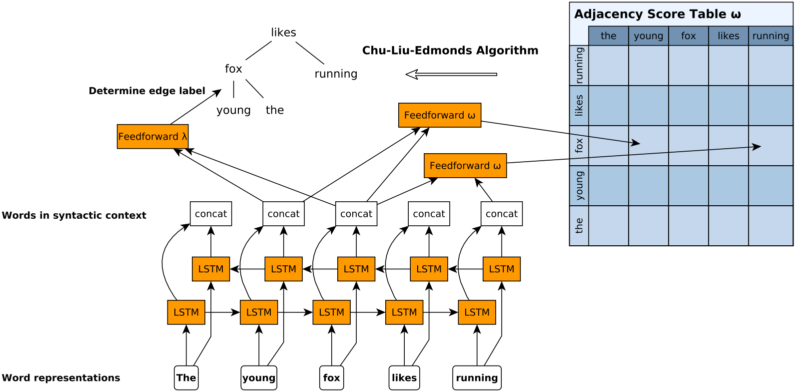


Corpora

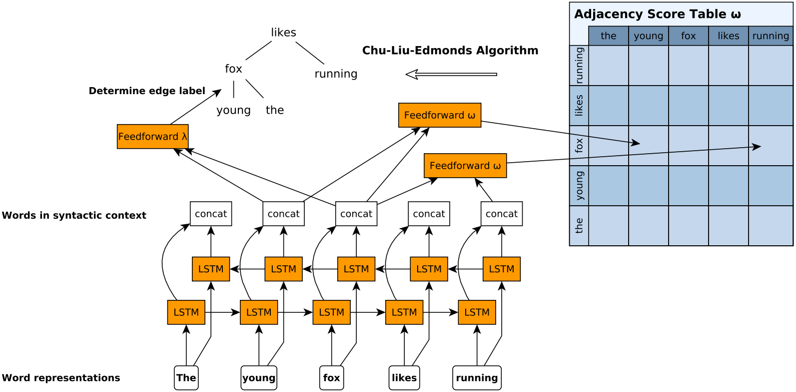
Approach evaluated on the following English corpora:

- PTB ($\approx 35,000$ sentences, news text) annotated with DM, PAS, PSD and EDS
- AMR 2015: $\approx 19,000$ sentences
- AMR 2017: $\approx 39,000$ sentences (news text, blog articles, online discussions)
- AMR 2015 $\subseteq$ AMR 2017

A Recent Dependency Parser (Kiperwasser and Goldberg, 2016)

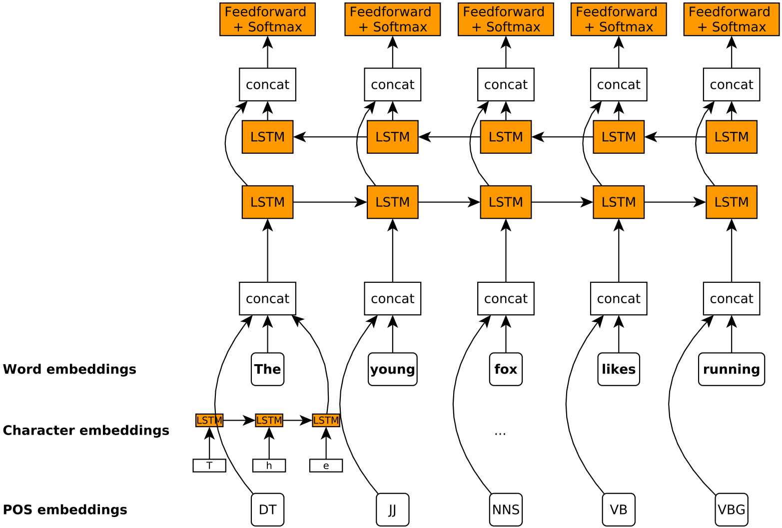


A Recent Dependency Parser (Kiperwasser and Goldberg, 2016)



Training objective: Push the score of the correct edges higher than (wrong) alternatives

Supertagging (similar to Lewis et al. (2016))

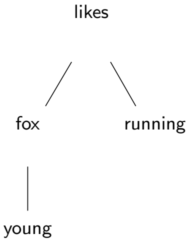


For each token and supertag, the NN gives us a score
$\theta(w_i, s) = \log P(w_i \text{ has supertag } s | w_1, \ldots, w_n)$

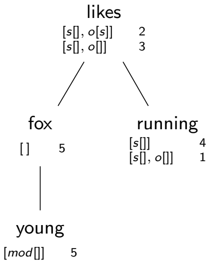
Fixed-Tree Decoding Algorithm

- bottom-up dynamic programming algorithm (similar to Viterbi)
- runtime complexity $\mathcal{O}(2^d \cdot n \cdot d)$ where d is maximal number of children in tree, n is size of the tree
- core idea: efficiently go through all AM terms that transform to the fixed tree, check types and manage scores

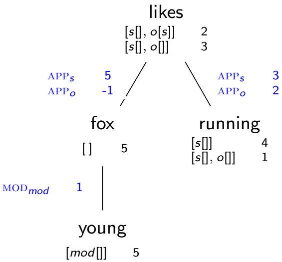
Fixed-Tree Decoding Algorithm



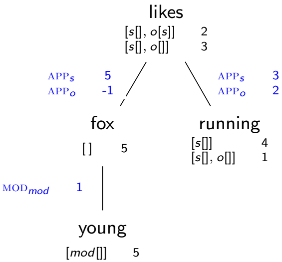
Fixed-Tree Decoding Algorithm

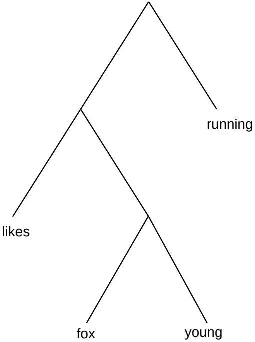


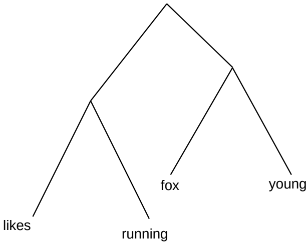
Fixed-Tree Decoding Algorithm



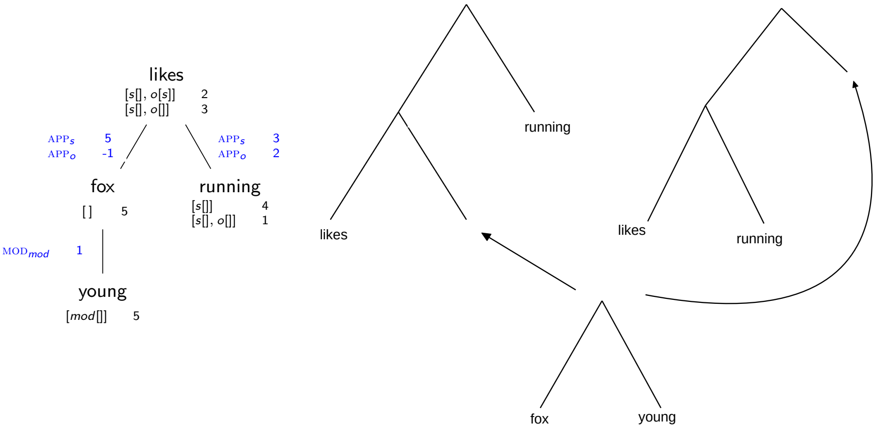
Fixed-Tree Decoding Algorithm



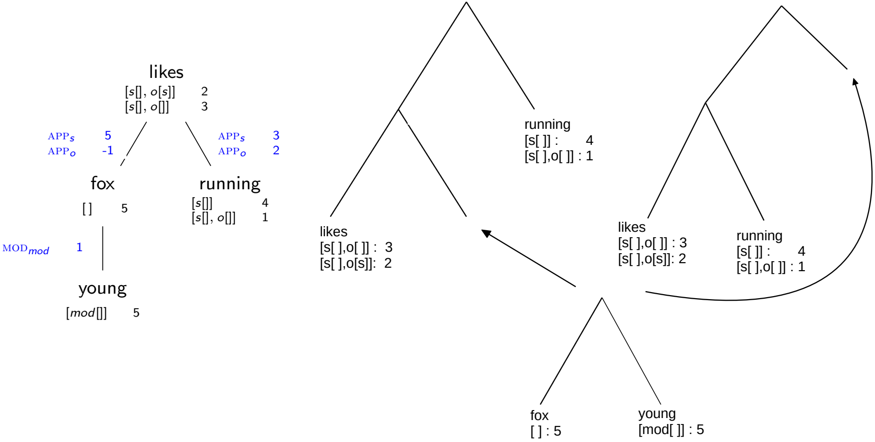




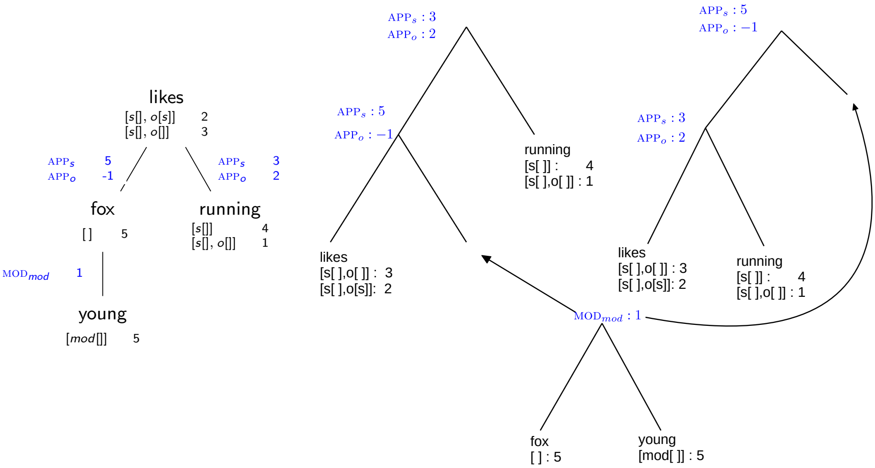
Fixed-Tree Decoding Algorithm



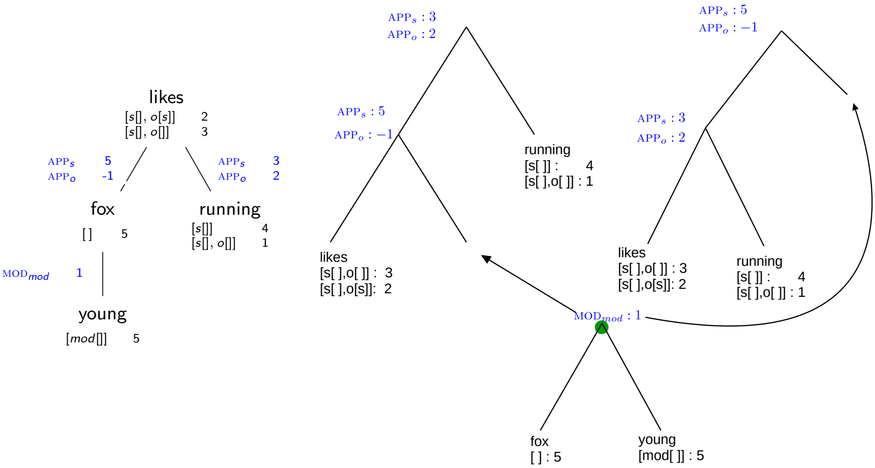
Fixed-Tree Decoding Algorithm



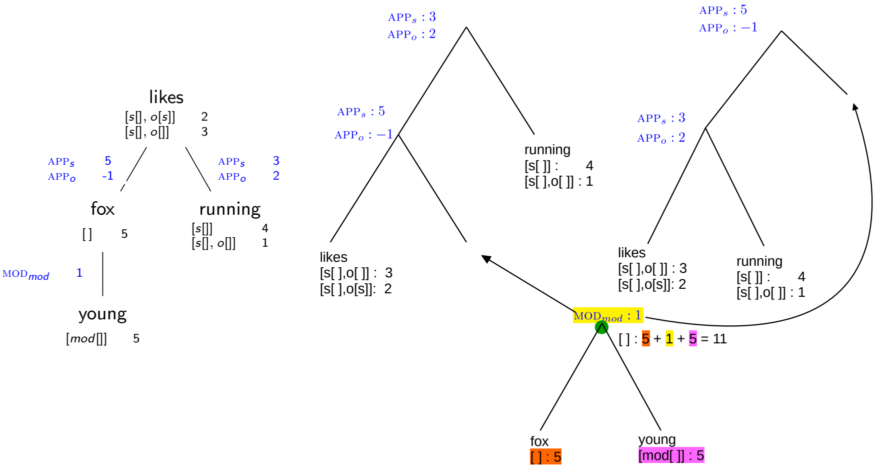
Fixed-Tree Decoding Algorithm



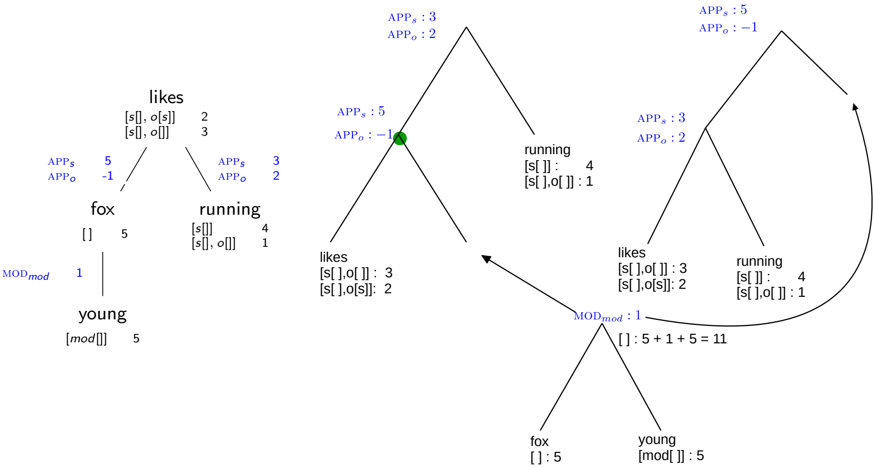
Fixed-Tree Decoding Algorithm



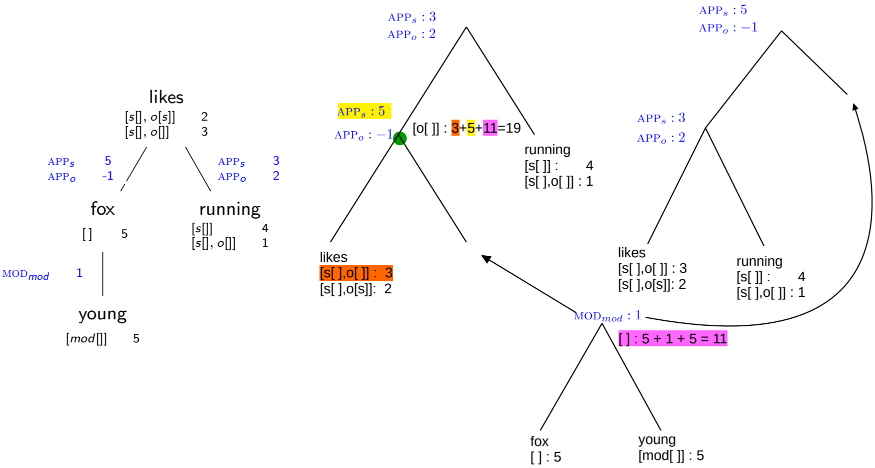
Fixed-Tree Decoding Algorithm



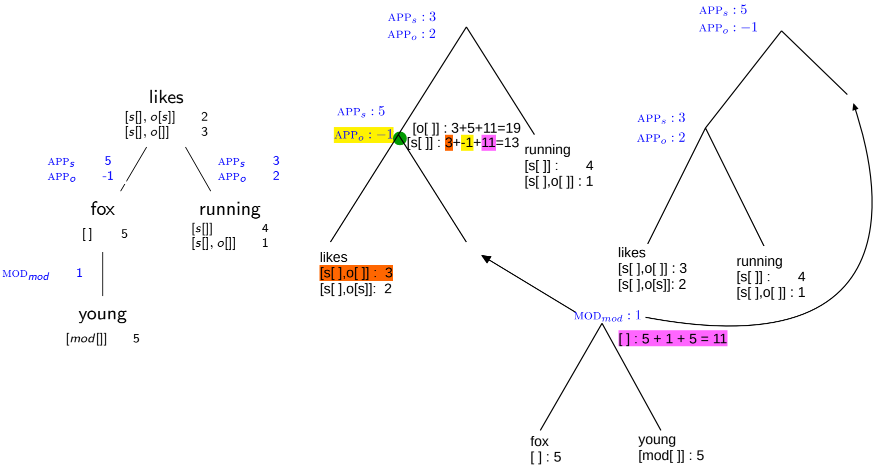
Fixed-Tree Decoding Algorithm



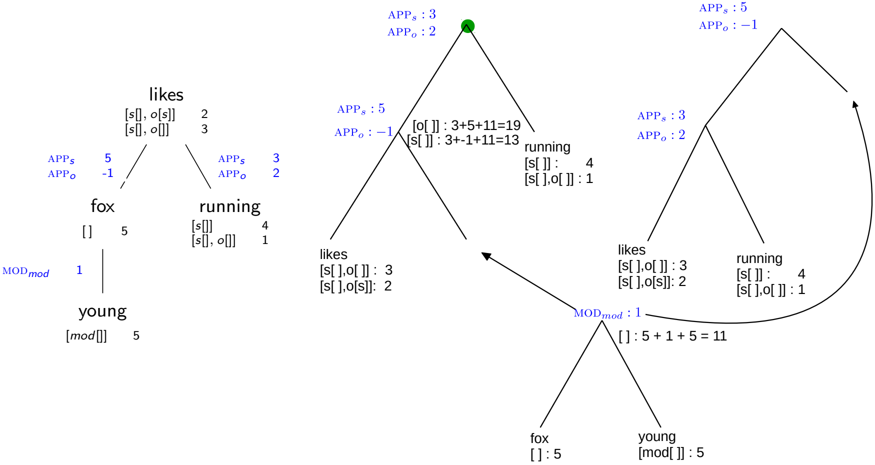
Fixed-Tree Decoding Algorithm



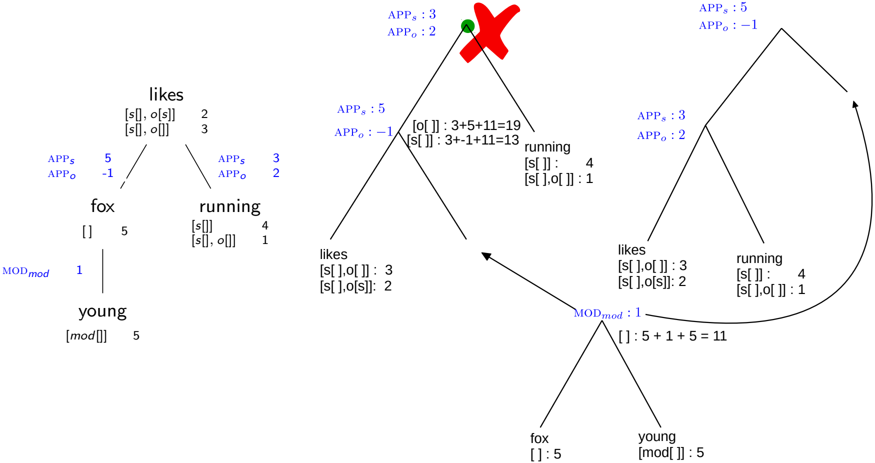
Fixed-Tree Decoding Algorithm



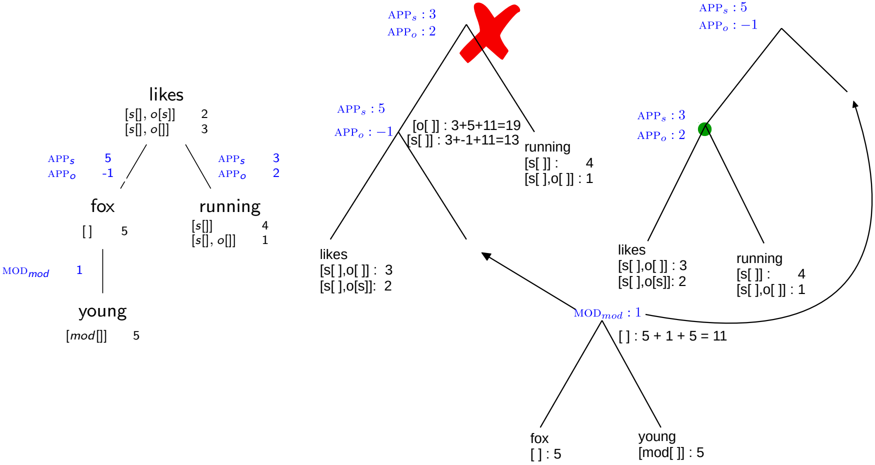
Fixed-Tree Decoding Algorithm



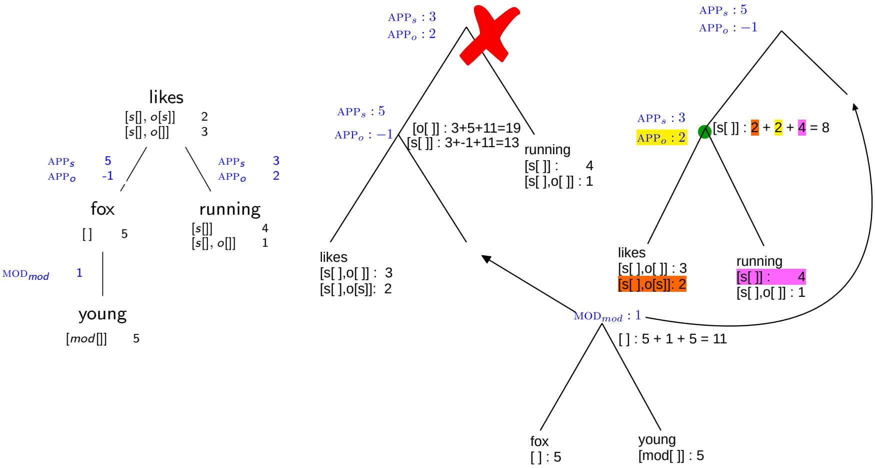
Fixed-Tree Decoding Algorithm



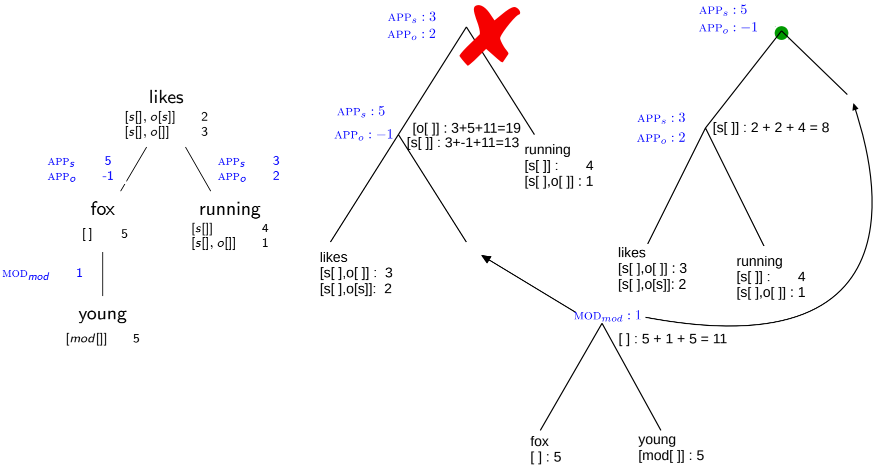
Fixed-Tree Decoding Algorithm



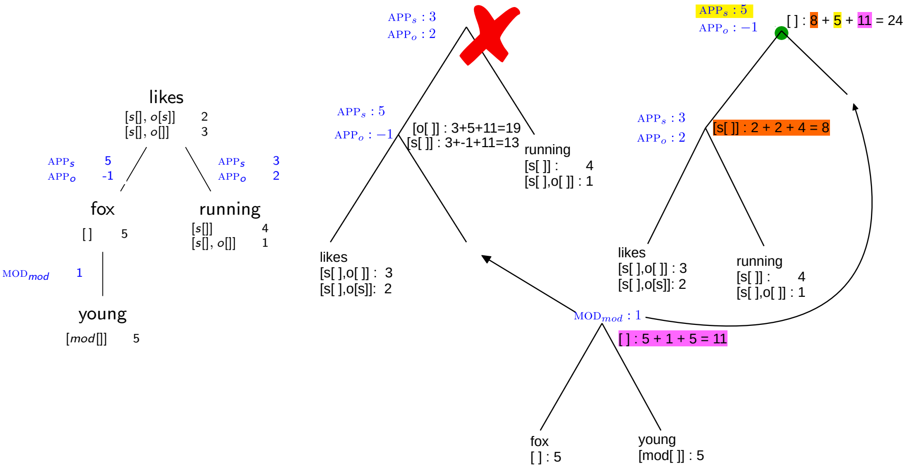
Fixed-Tree Decoding Algorithm



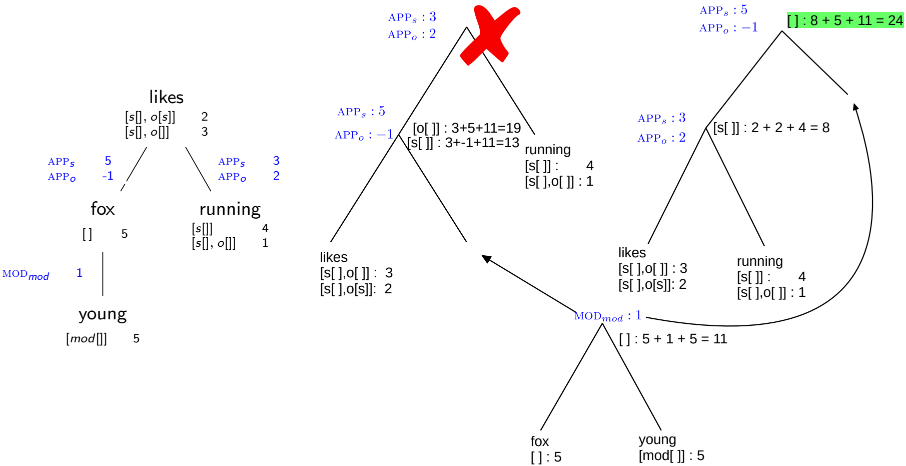
Fixed-Tree Decoding Algorithm



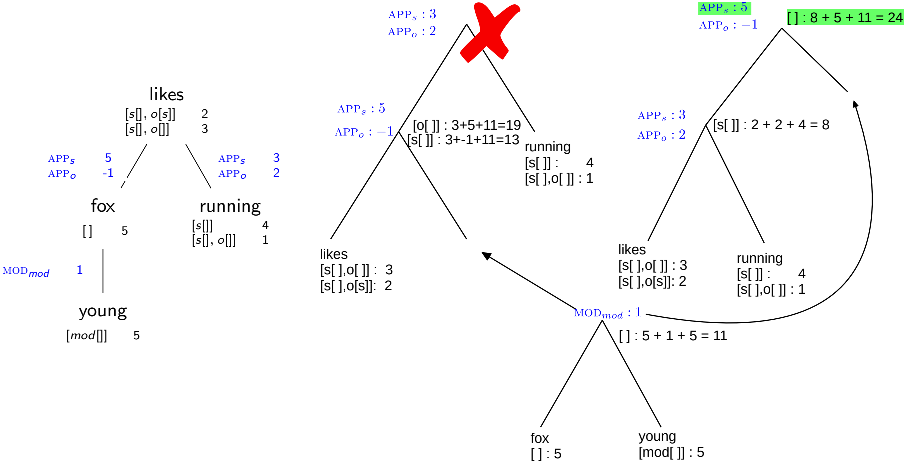
Fixed-Tree Decoding Algorithm



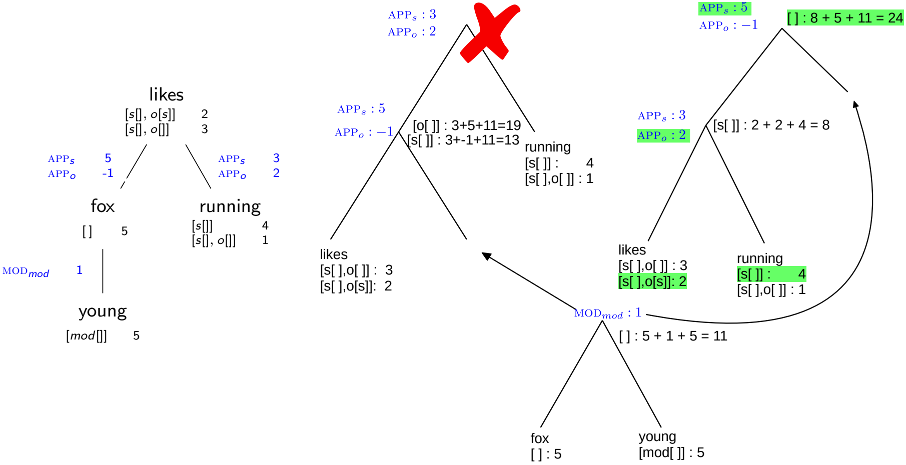
Fixed-Tree Decoding Algorithm



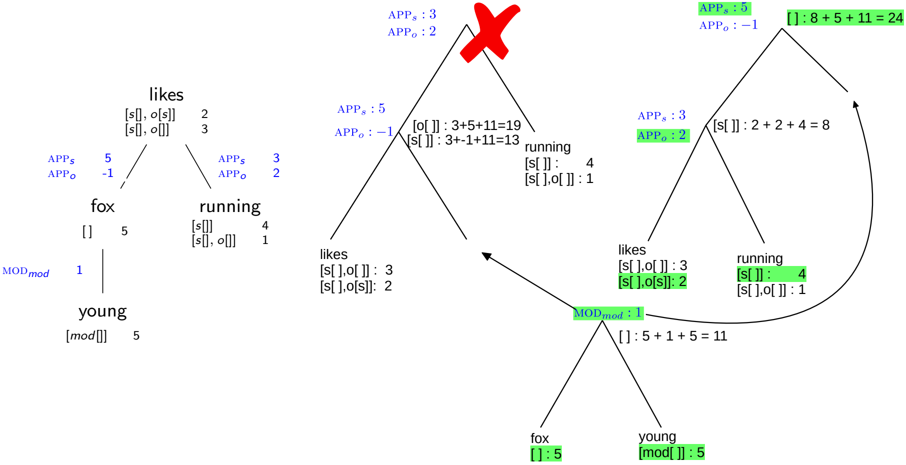
Fixed-Tree Decoding Algorithm



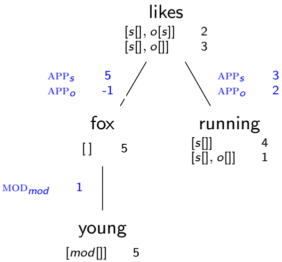
Fixed-Tree Decoding Algorithm

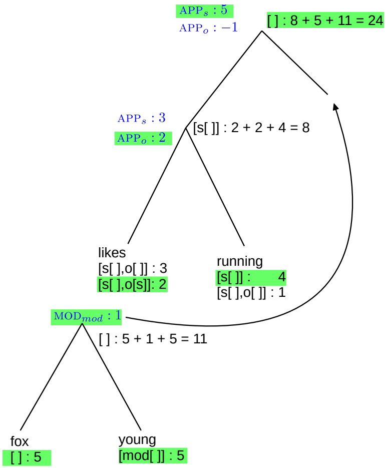


Fixed-Tree Decoding Algorithm

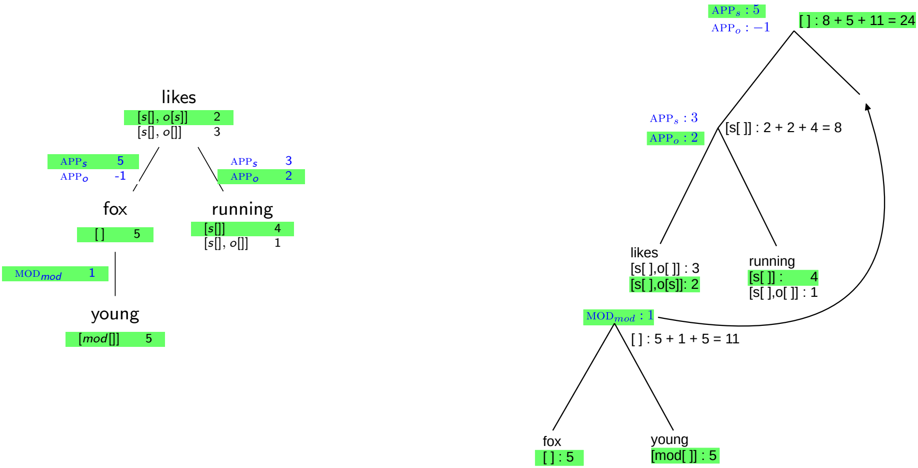


Fixed-Tree Decoding Algorithm

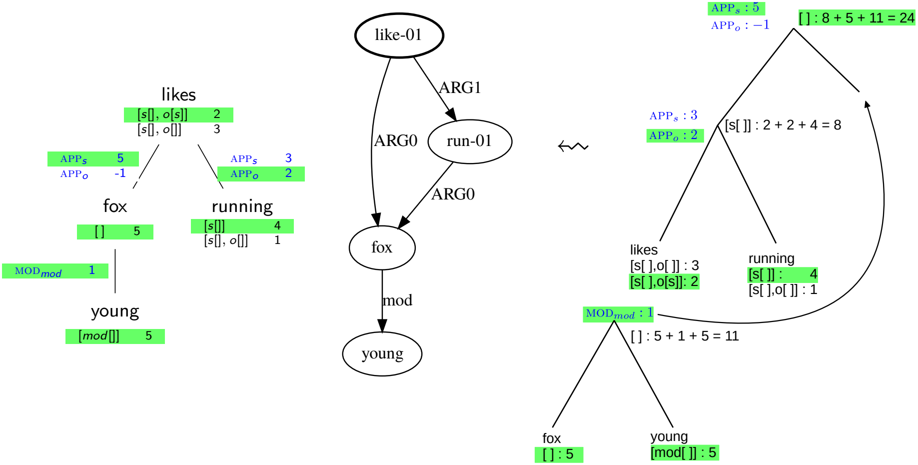




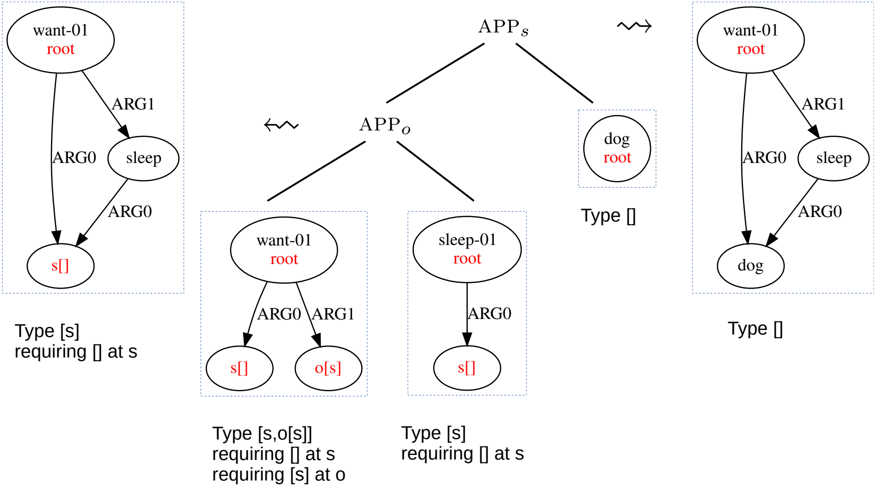
Fixed-Tree Decoding Algorithm



Fixed-Tree Decoding Algorithm



APP and Reentrancy

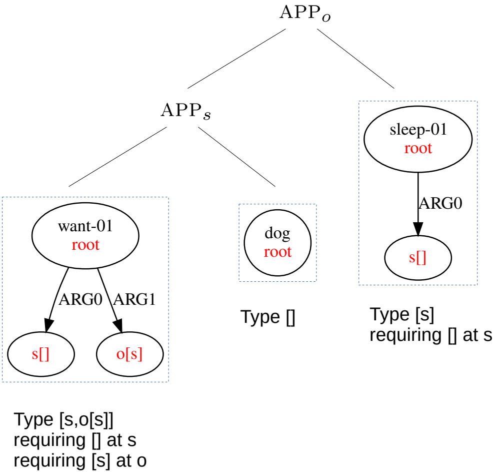


The dog wants to sleep

Constant for *want* enforces reentrancy!

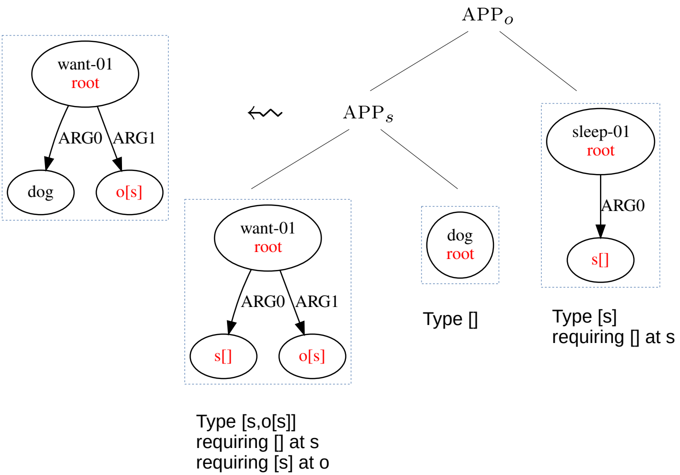
Can we first fill in the subject?

No!

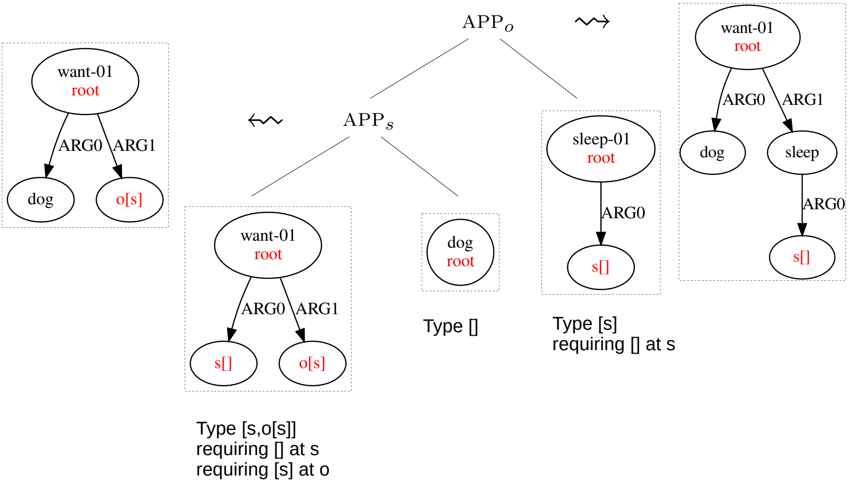


APP$_o$
APP$_s$
want-01
root
ARG0
ARG1
s[]
o[s]
dog
root
sleep-01
root
ARG0
s[]
Type []
Type [s]
requiring [] at s
Type [s,o[s]]
requiring [] at s
requiring [s] at o

No!

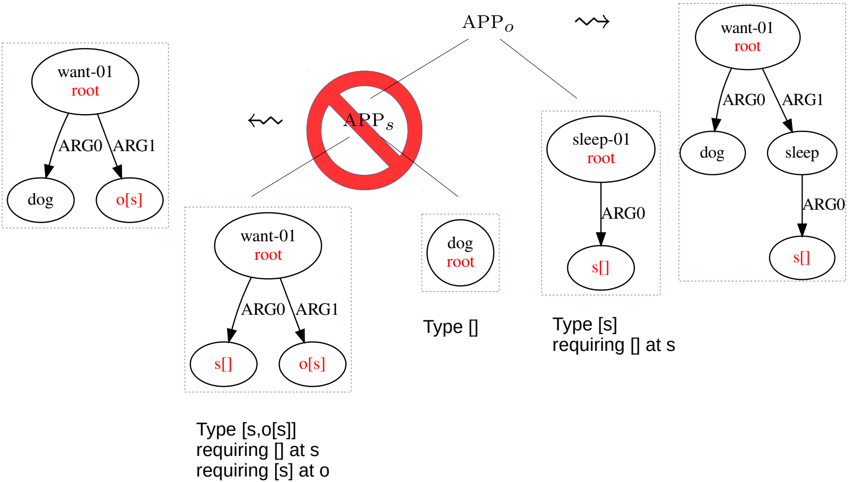

want-01
root
ARG0
ARG1
dog
o[s]
APP$_o$
APP$_s$
want-01
root
ARG0
ARG1
s[]
o[s]
dog
root
sleep-01
root
ARG0
s[]
Type []
Type [s]
requiring [] at s
Type [s,o[s]]
requiring [] at s
requiring [s] at o

No!



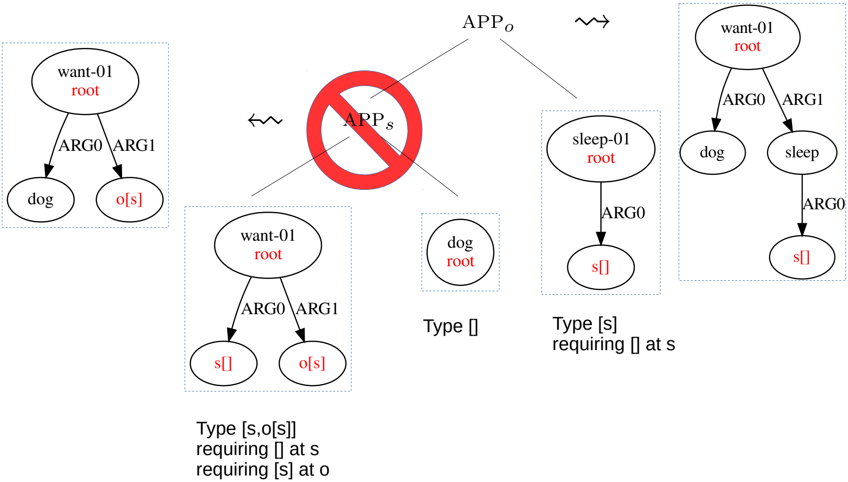
APP$_o$
APP$_s$
want-01
root
ARG0
ARG1
dog
o[s]
s[]
dog
root
sleep-01
root
ARG0
s[]
want-01
root
ARG0
ARG1
dog
sleep
ARG0
s[]
Type [s,o[s]]
requiring [] at s
requiring [s] at o
Type []
Type [s]
requiring [] at s

No!



APP_s must not come first because s is in the requirement of o-source!

No!



APP_s must not come first because s is in the requirement of o-source!
Conclusion: Order of application is not arbitrary.