AMR dependency parsing with a typed semantic algebra

Jonas Groschwitz, Matthias Lindemann,
Meaghan Fowlie, Mark Johnson, Alexander Koller

March 2018

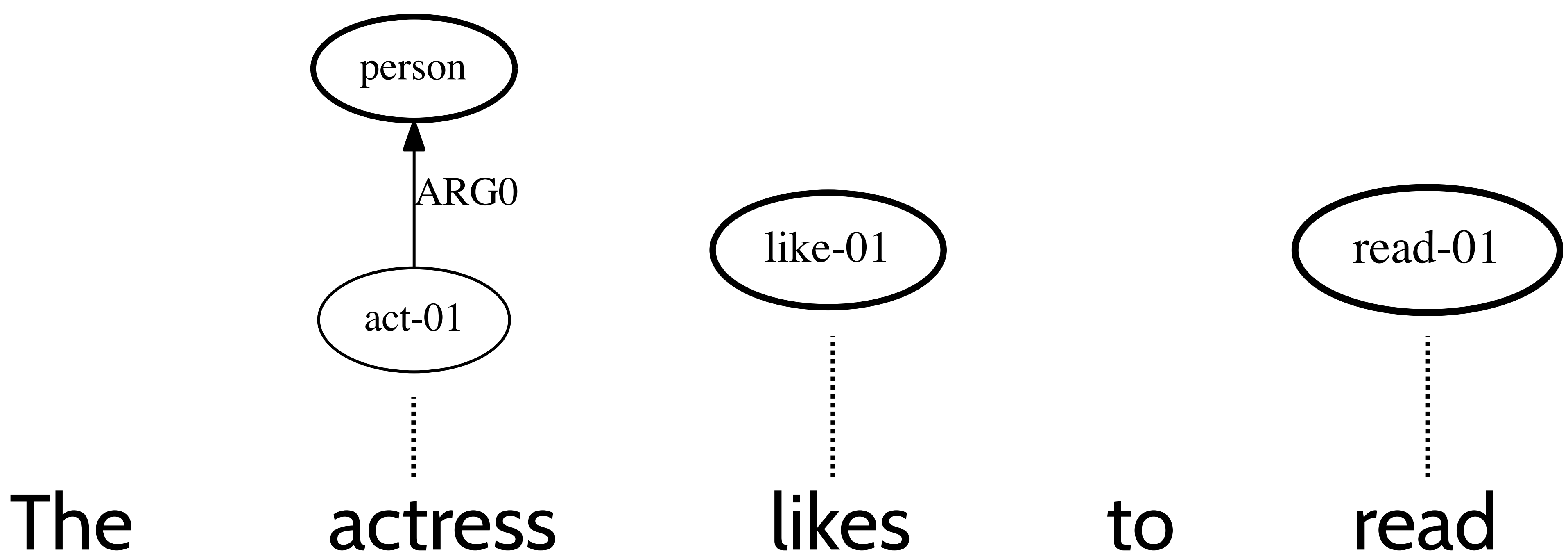
Introduction

The actress likes to read

Introduction

A common approach to AMR parsing:

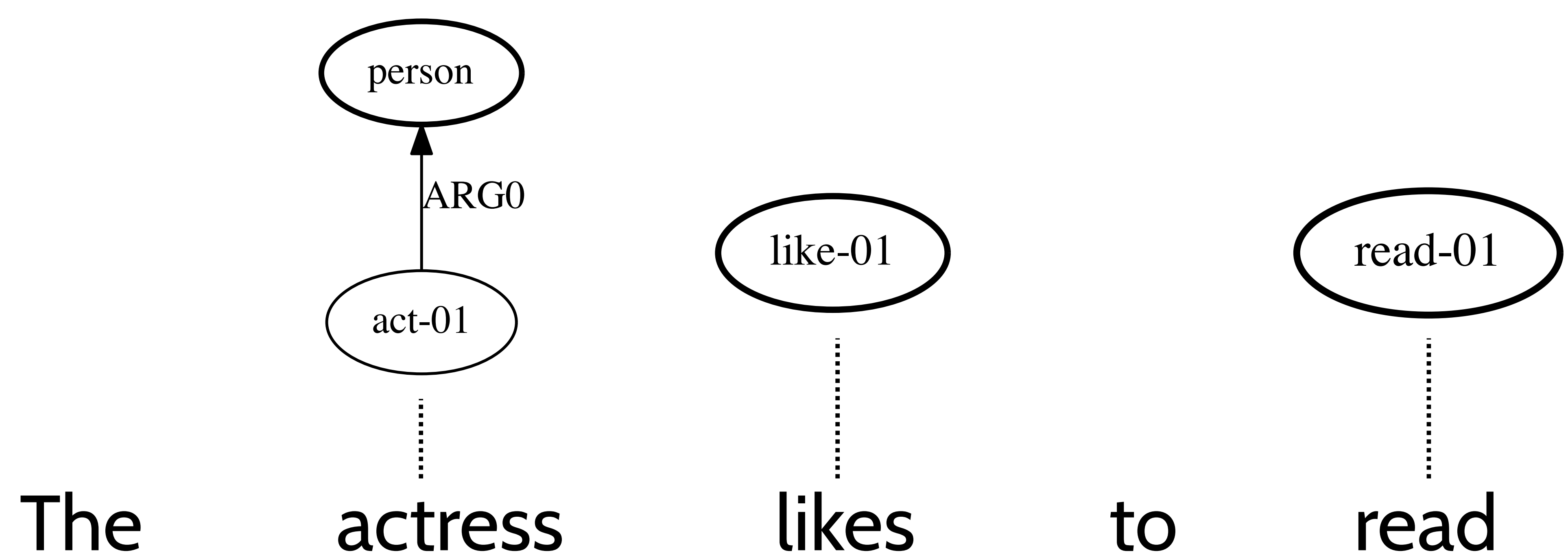
1. Predict graph fragments for words



Introduction

A common approach to AMR parsing:

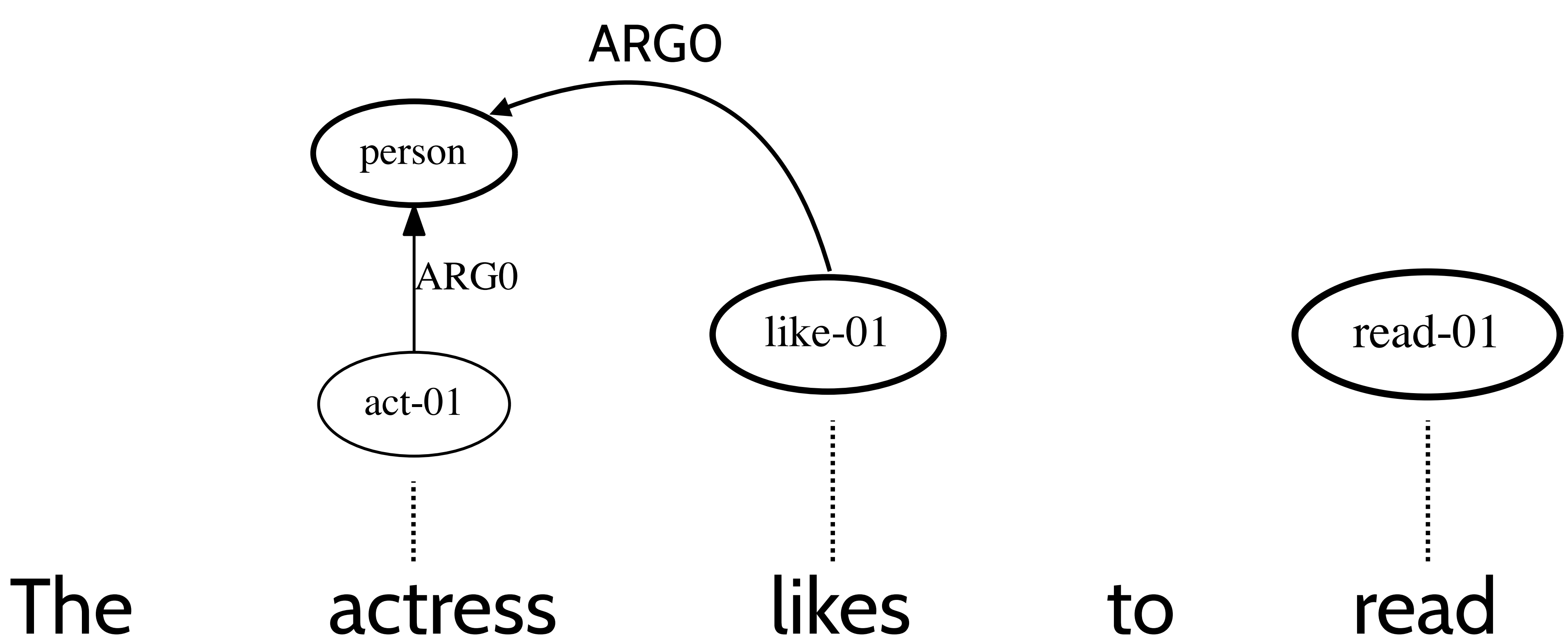
1. Predict graph fragments for words
2. Predict relations (edges) between the graph fragments



Introduction

A common approach to AMR parsing:

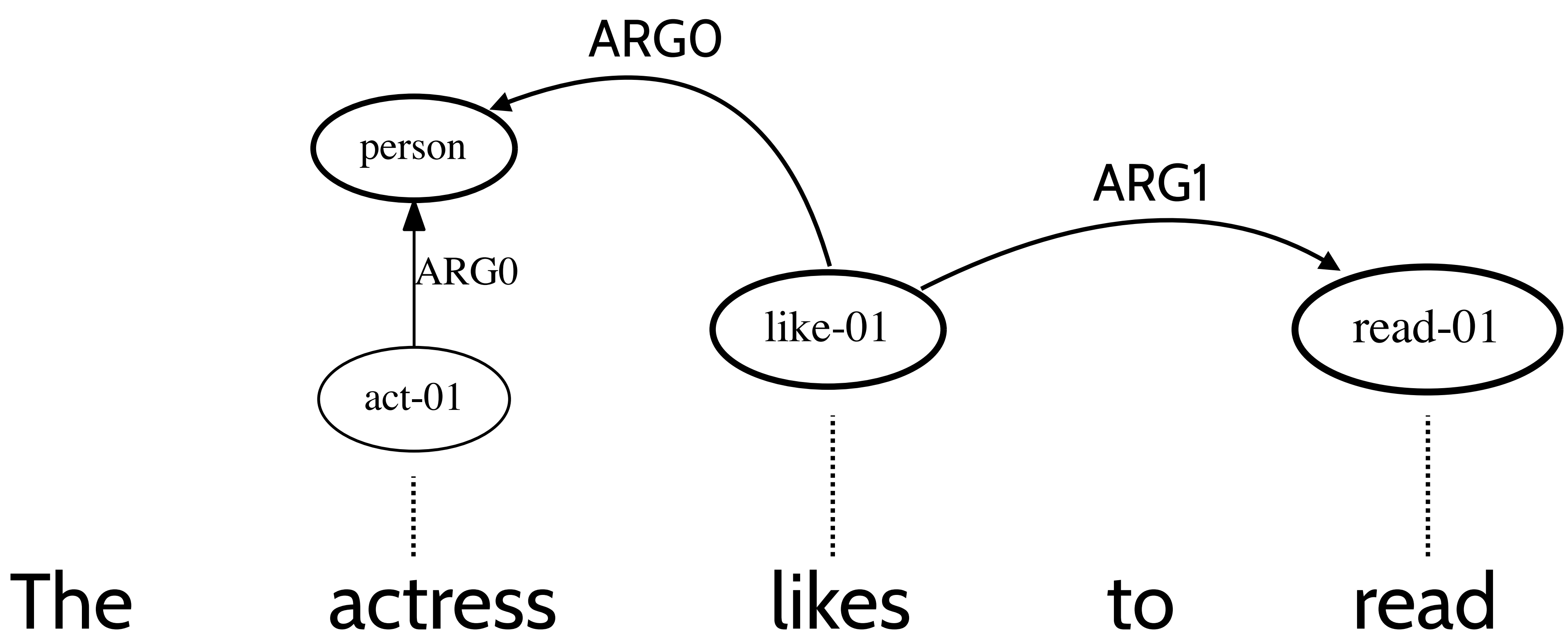
1. Predict graph fragments for words
2. Predict relations (edges) between the graph fragments



Introduction

A common approach to AMR parsing:

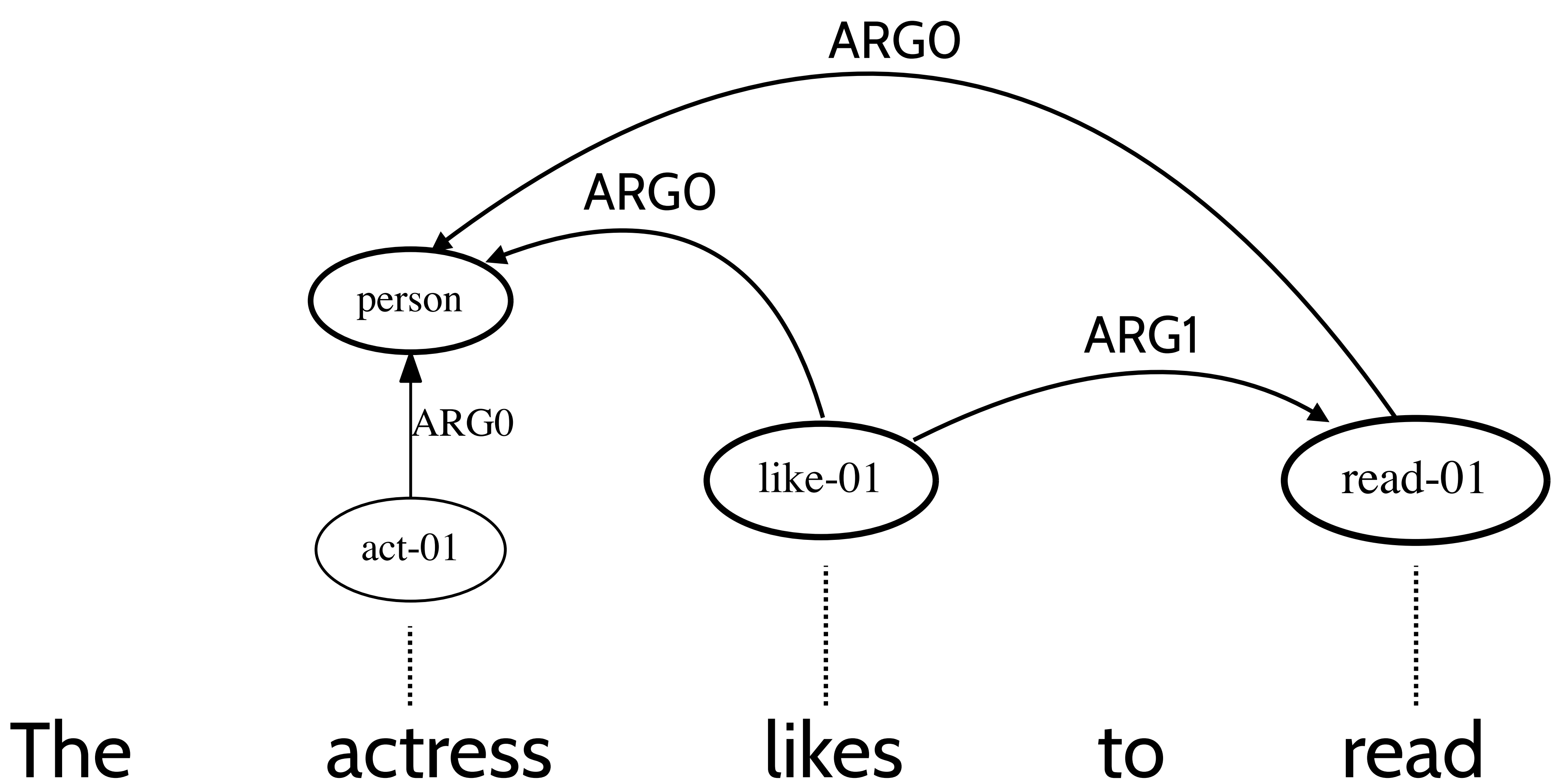
1. Predict graph fragments for words
2. Predict relations (edges) between the graph fragments



Introduction

A common approach to AMR parsing:

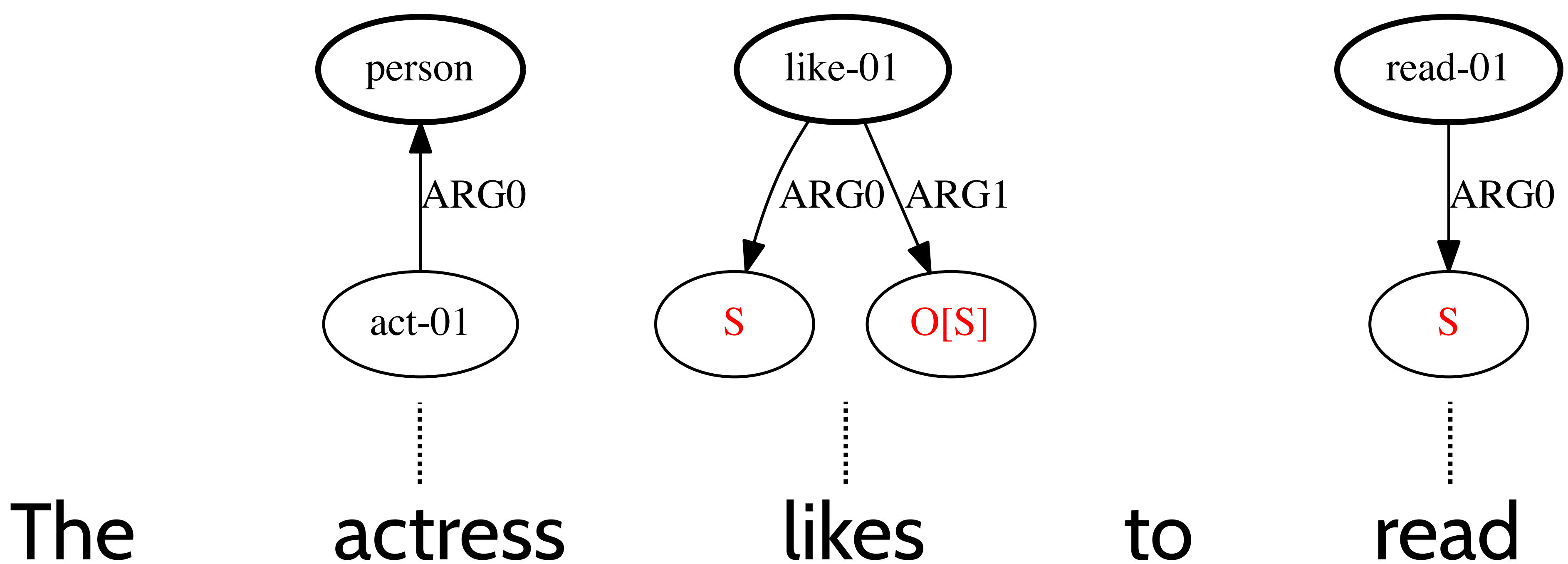
1. Predict graph fragments for words
2. Predict relations (edges) between the graph fragments



Introduction

Our Approach: AM dependency tree.

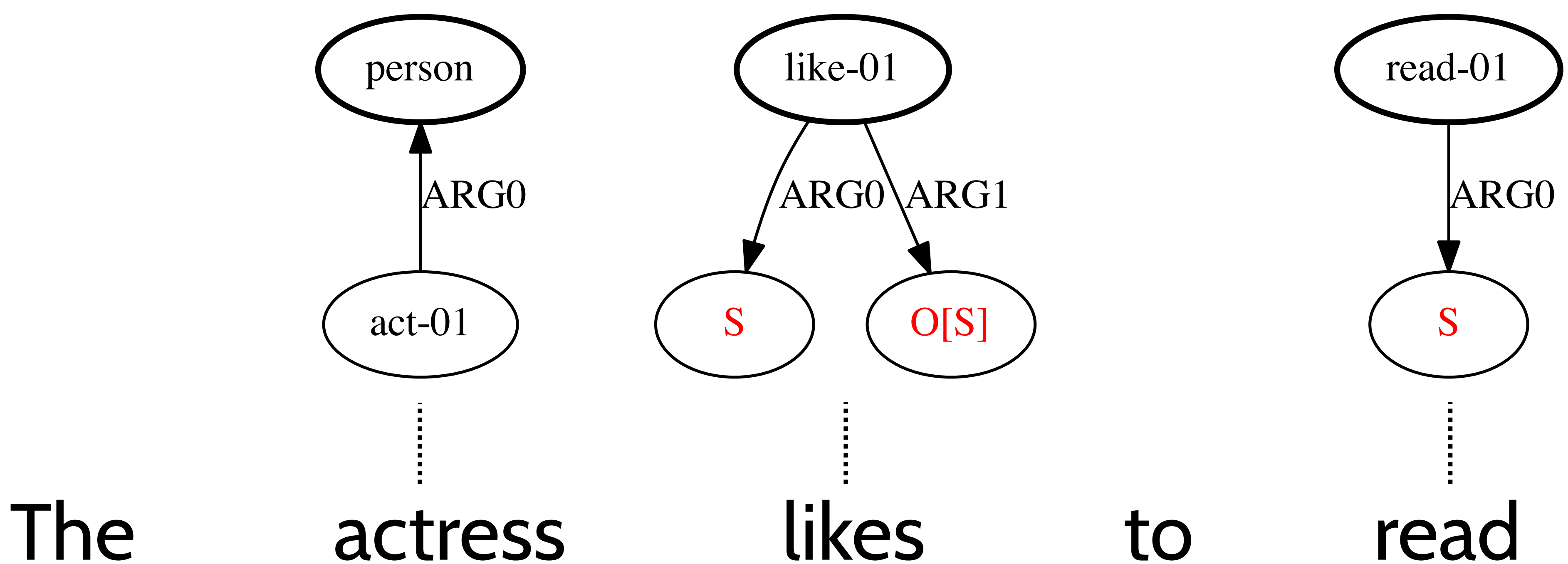
1. Predict as-graphs for words



Introduction

Our Approach: AM dependency tree.

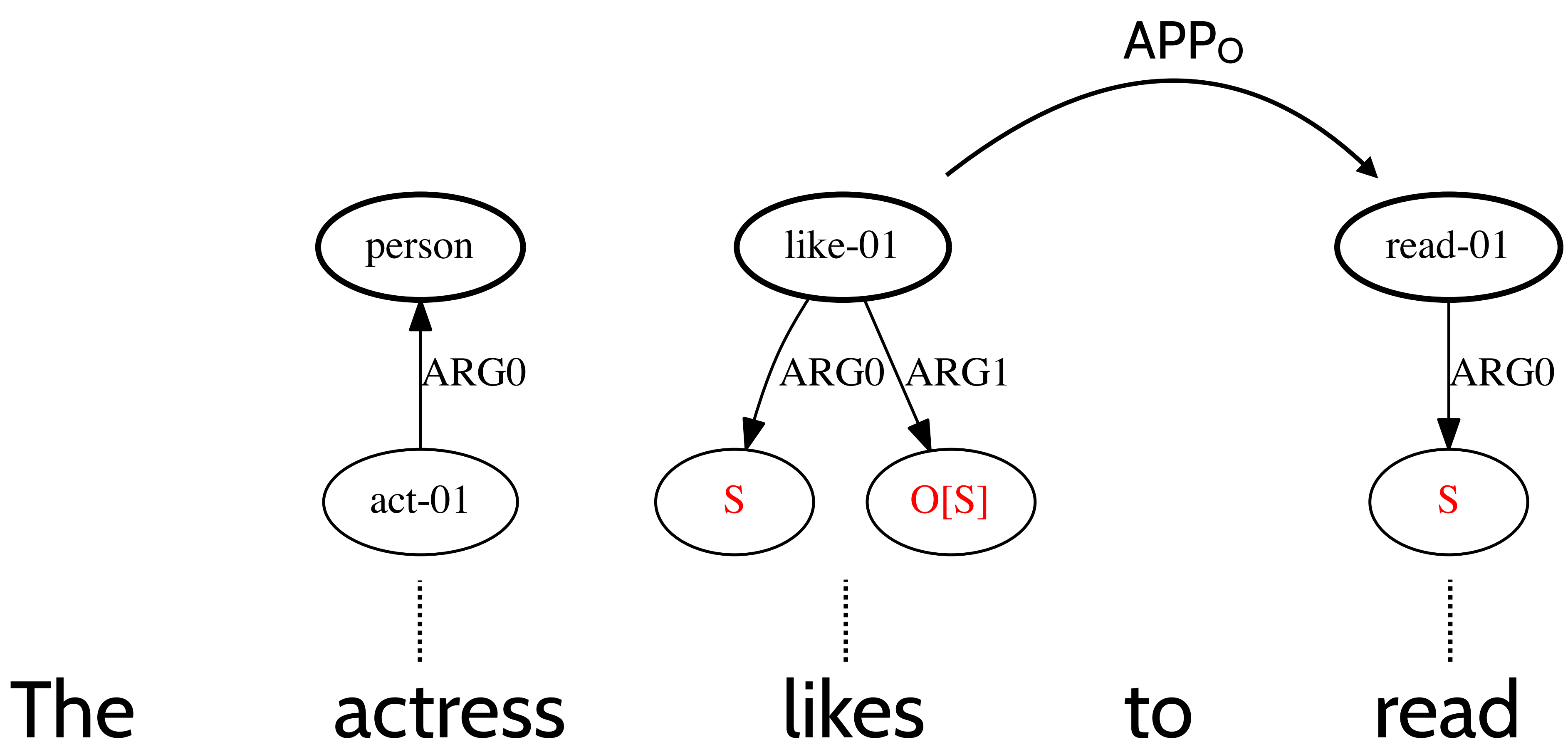
1. Predict as-graphs for words
2. Predict operations between them



Introduction

Our Approach: AM dependency tree.

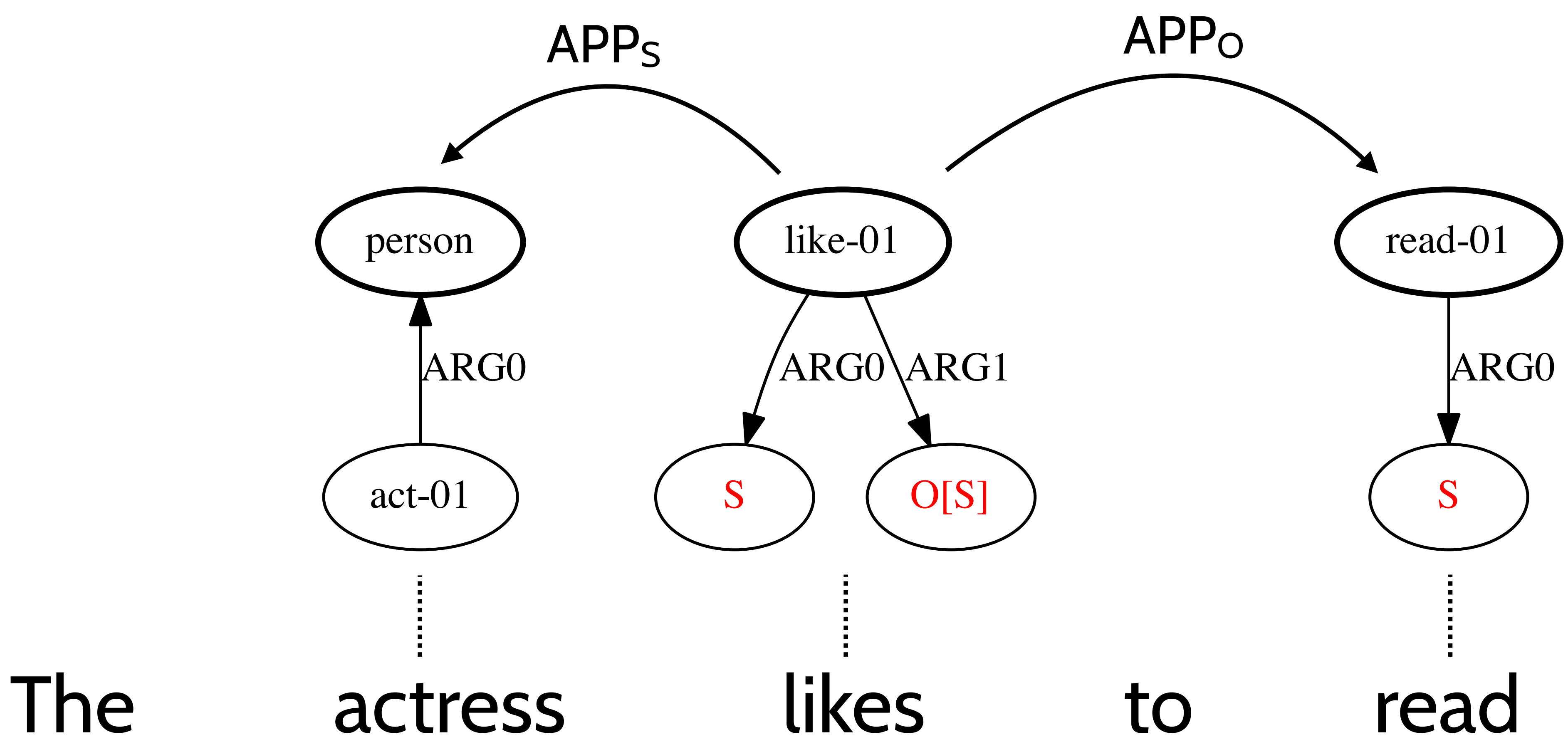
1. Predict as-graphs for words
2. Predict operations between them



Introduction

Our Approach: AM dependency tree.

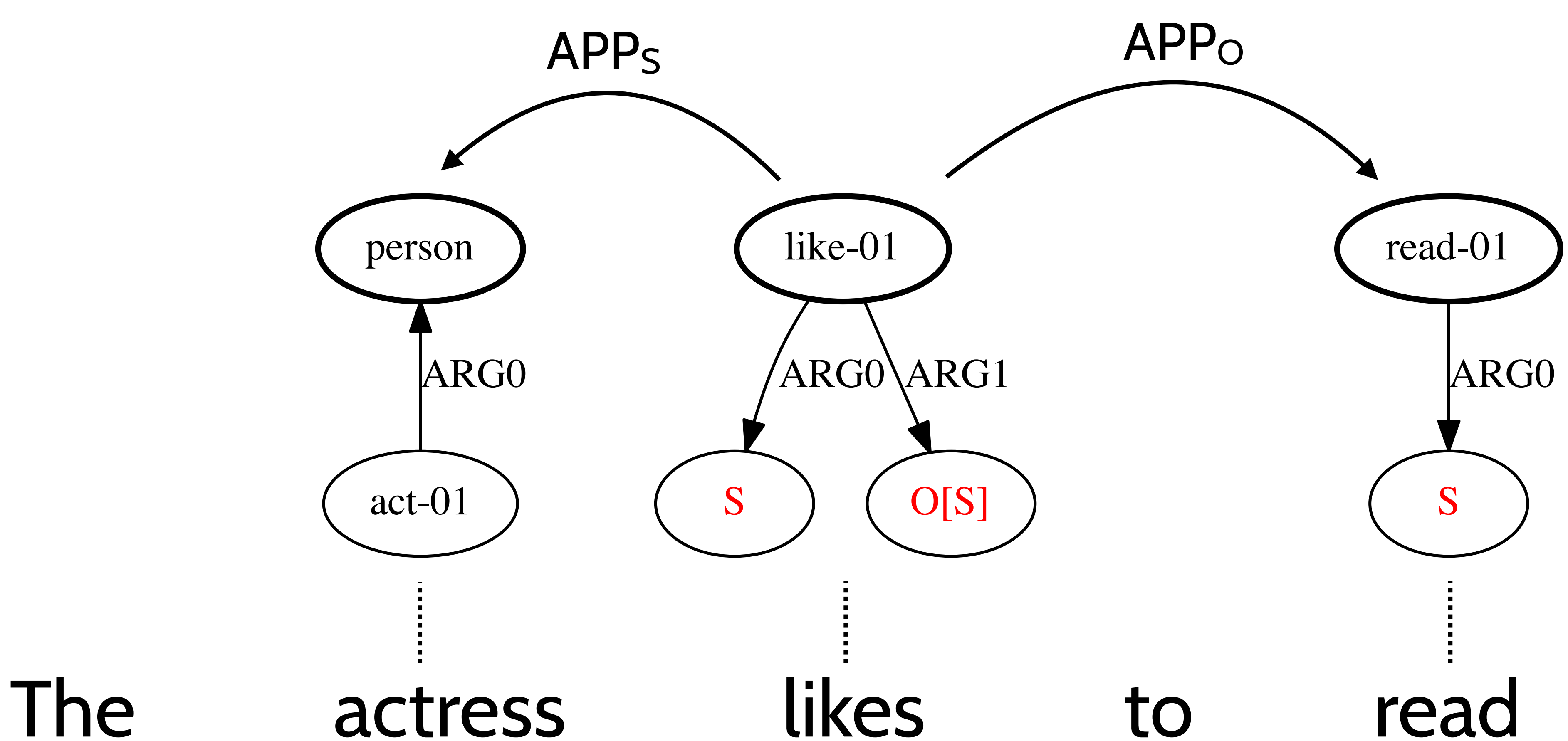
1. Predict as-graphs for words
2. Predict operations between them



Introduction

Advantages:

1. We predict a tree, not a graph, i.e. we an use dependency parsing methods.
2. Can put our linguistic knowledge into graph types, to guide our parser.
3. AM dependency tree is a compositional structure. Can examine it from both engineering + linguistics perspective.



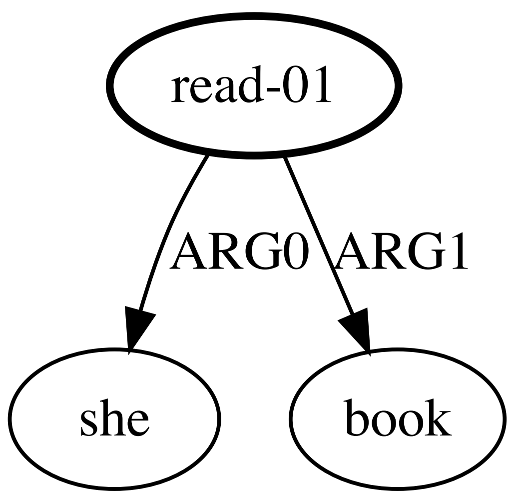
1. AM dependency trees (and why they make sense)
2. The parser in practice
3. Examples

1a. Towards AM Dependency Trees:

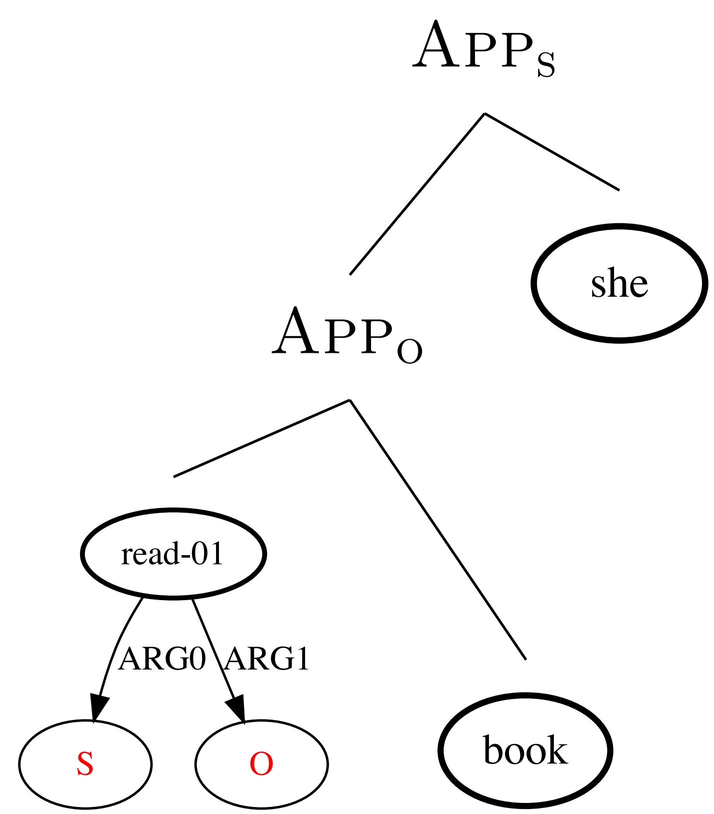
Indexed AM Terms

Indexed AM terms

Connect a given AM term with a sentence.

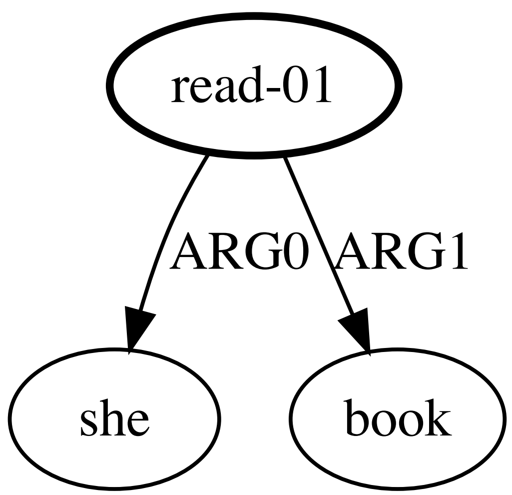


she$_0$ reads$_1$ a$_2$ book$_3$

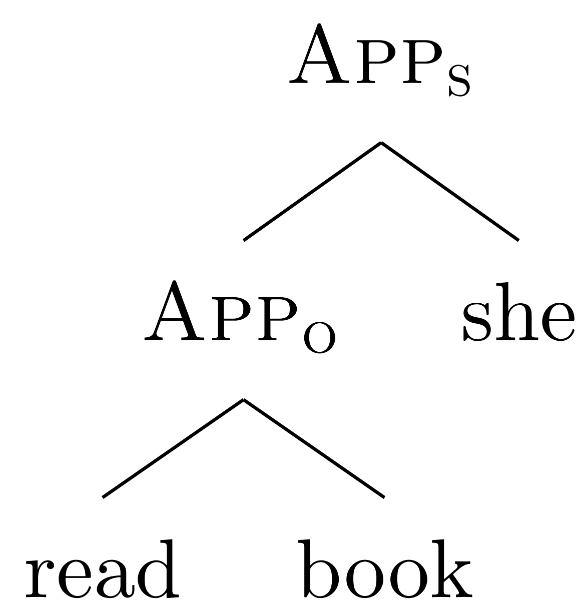


Indexed AM terms

Connect a given AM term with a sentence.



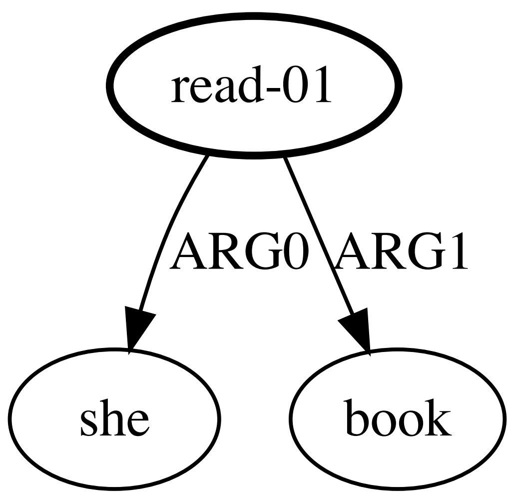
she$_0$ reads$_1$ a$_2$ book$_3$



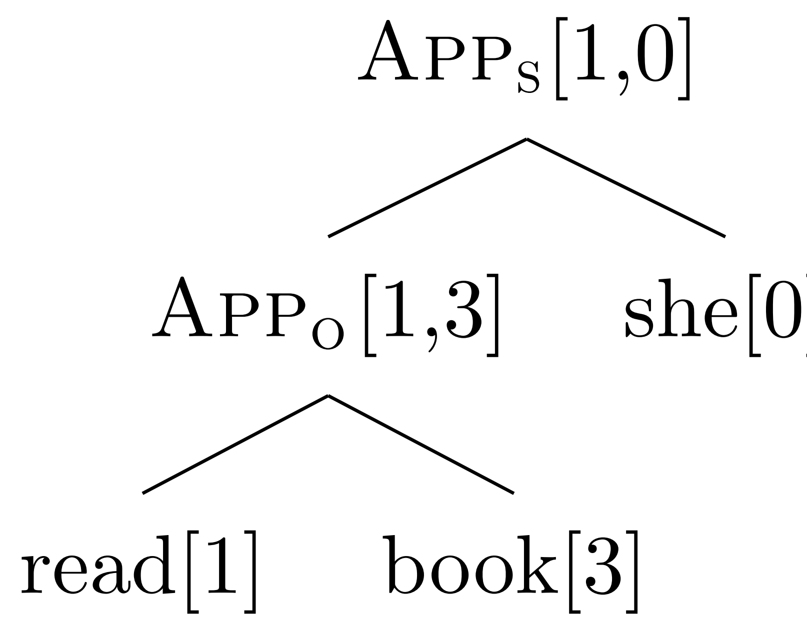
Indexed AM terms

Connect a given AM term with a sentence.

1. add indices to elementary graphs (essentially: alignments)
2. percolate indices upwards, mirroring the behaviour of the graph root (i.e. percolate the index of the left argument)



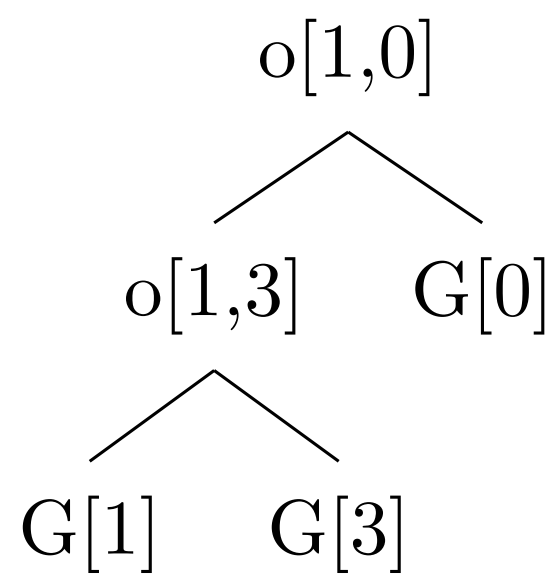
she$_0$ reads$_1$ a$_2$ book$_3$



Indexed AM terms

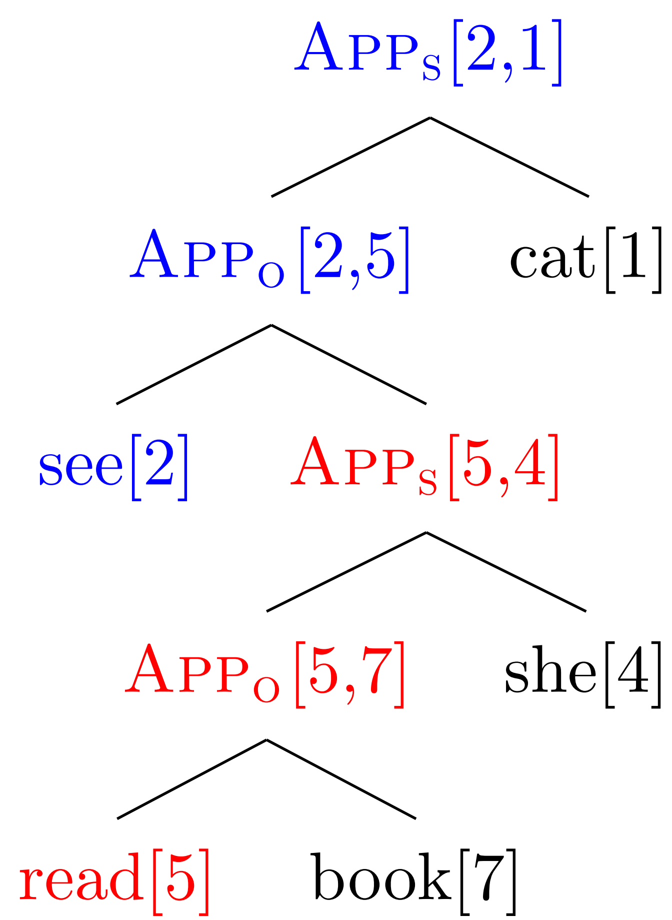
Connect a given AM term with a sentence.

1. add indices to elementary graphs (essentially: alignments)
2. percolate indices upwards, mirroring the behaviour of the graph root (i.e. percolate the index of the left argument)



Indexed AM terms

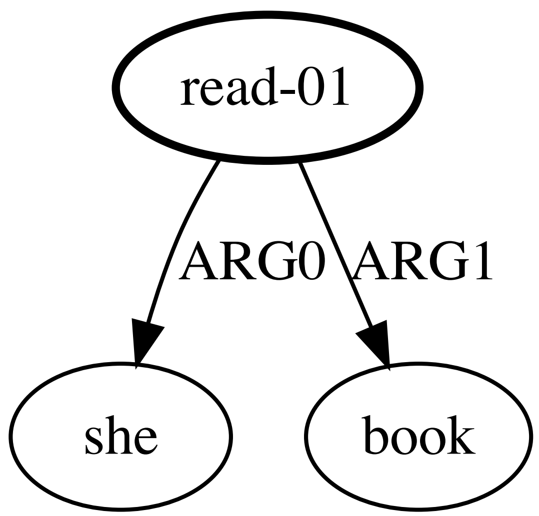
- An index persists on the left, until the corresponding root is consumed (index on the right). Sort of maximal projection.

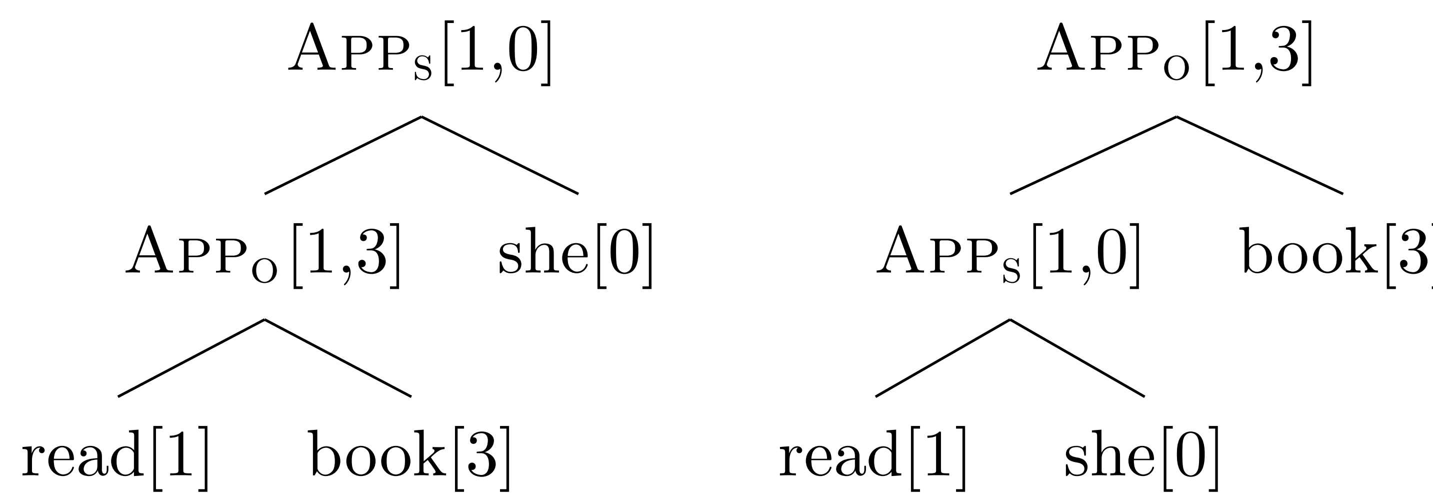


the$_0$ cat$_1$ sees$_2$ that$_3$ she$_4$ reads$_5$ a$_6$ book$_7$

Indexed AM terms

- changing operation order within a maximal projection does not change the outcome

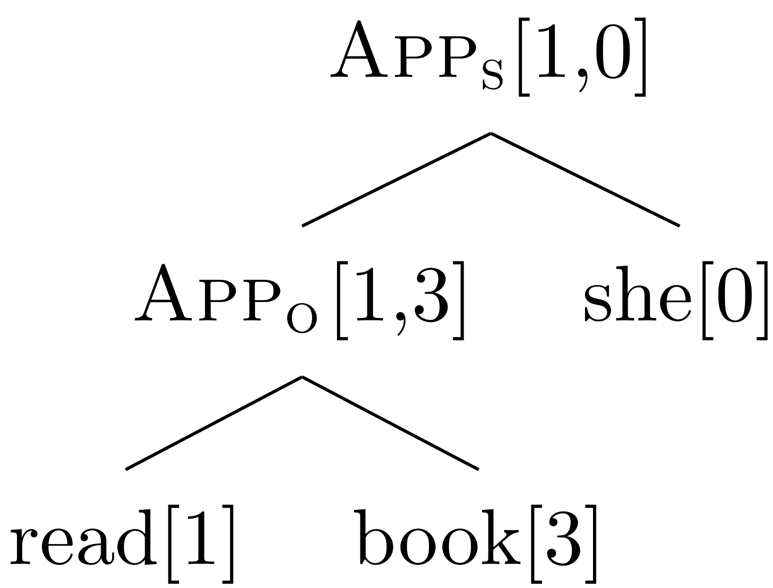


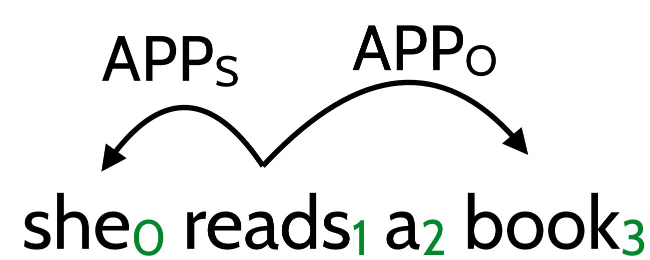


Indexed AM terms

We can see the operations as edges between words, by interpreting o[i,j] as an edge from i to j with label o.

Example:

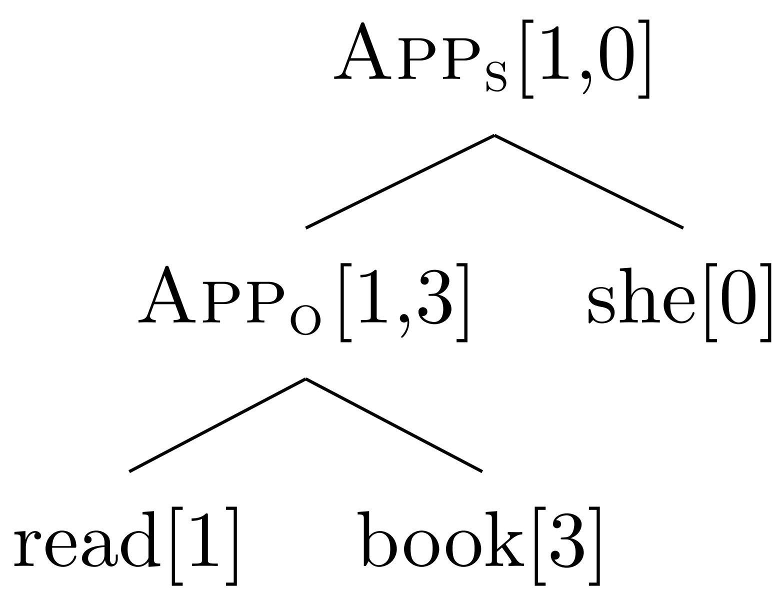


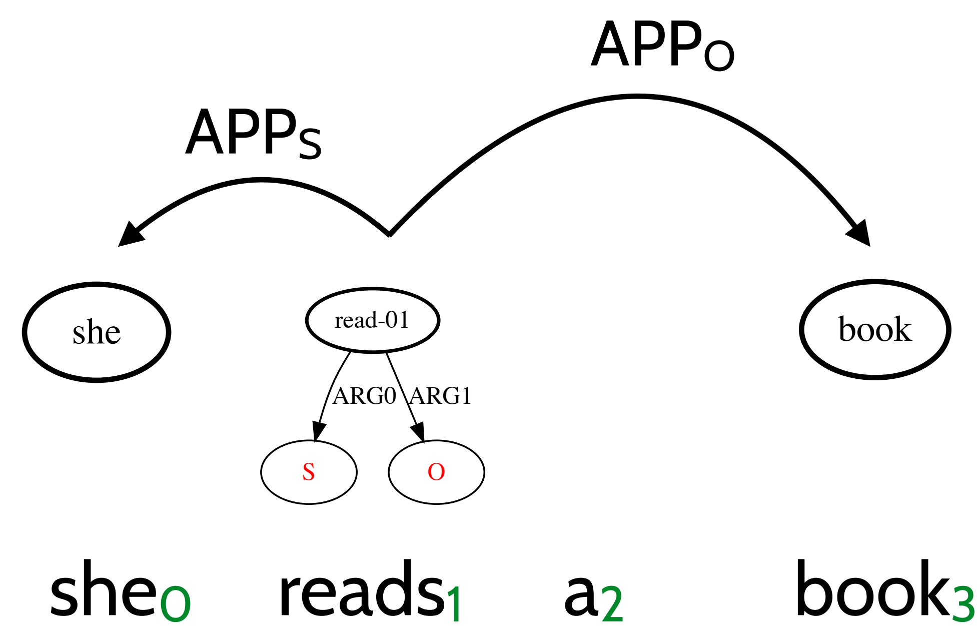


1b. AM dependency trees

AM dependency trees

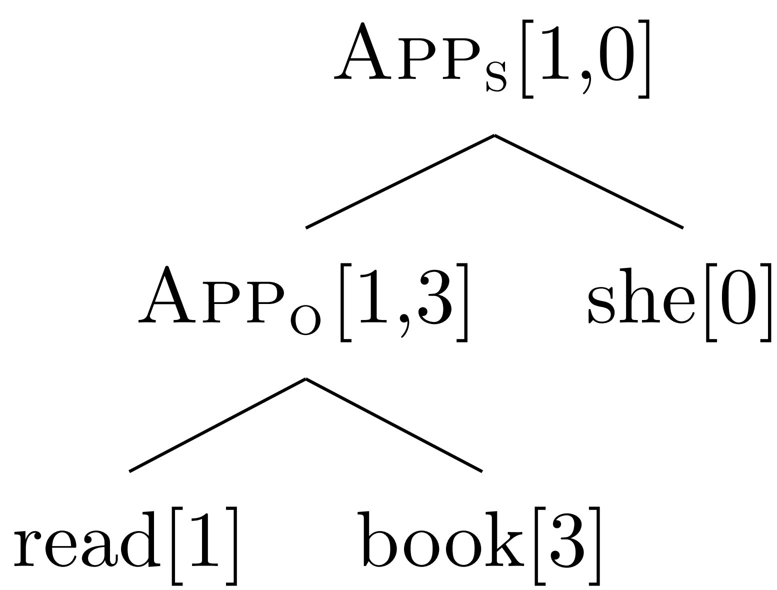
- Write an indexed AM term as a dependency tree. Operations are edges, nodes are elementary graphs per word.

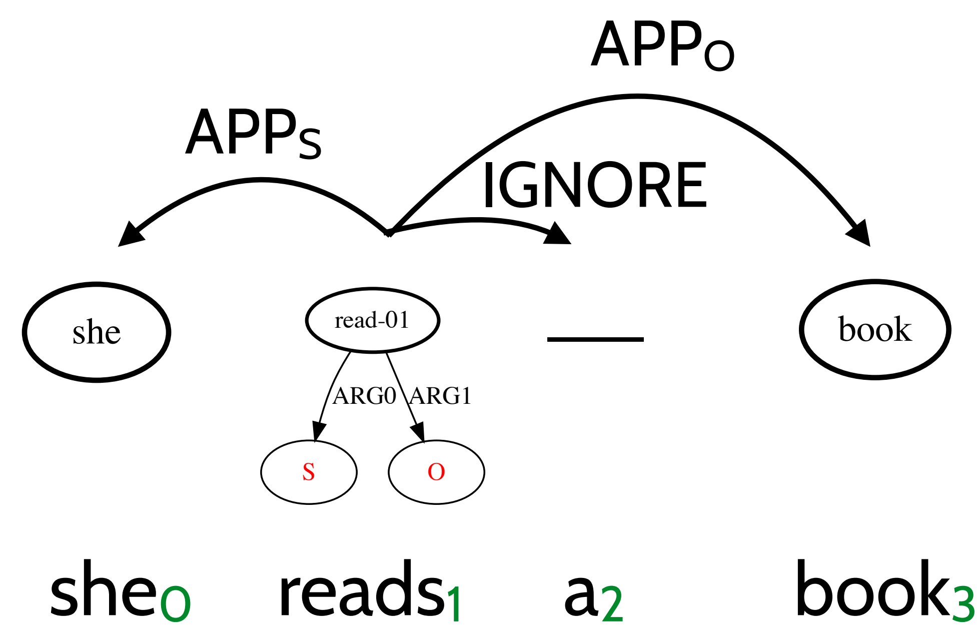




AM dependency trees

- Write an indexed AM term as a dependency tree. Operations are edges, nodes are elementary graphs per word.

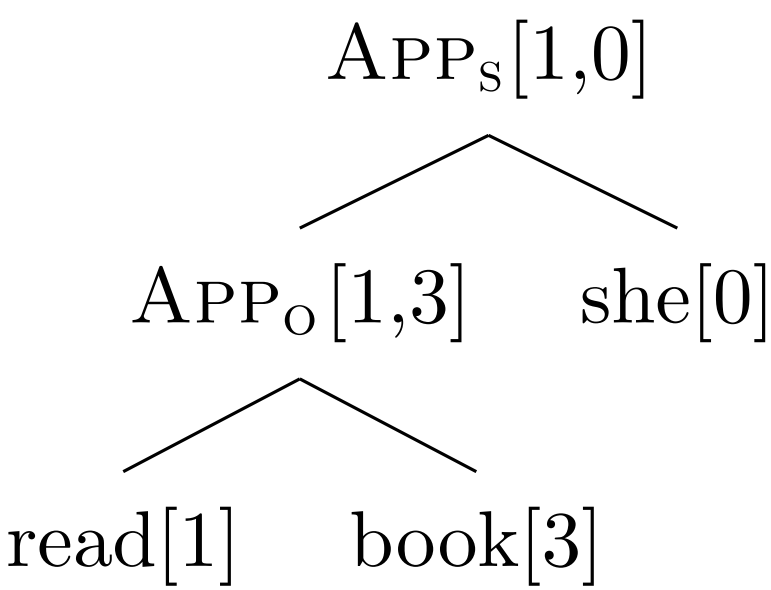


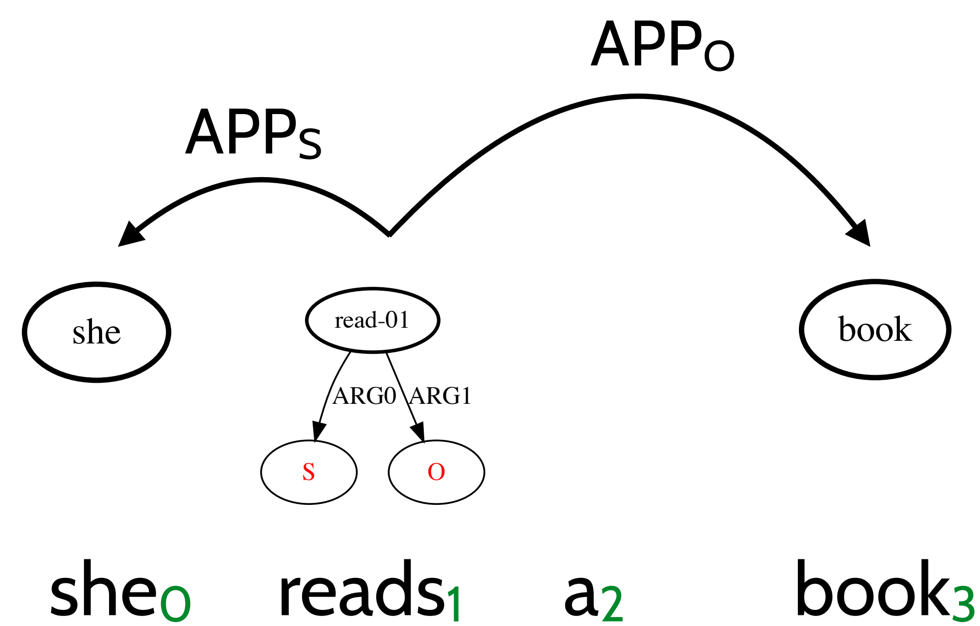


- add 'IGNORE' edges for words that are not represented in semantics. (won't use these in this talk)

AM dependency trees

- Write an indexed AM term as a dependency tree. Operations are edges, nodes are elementary graphs per word.



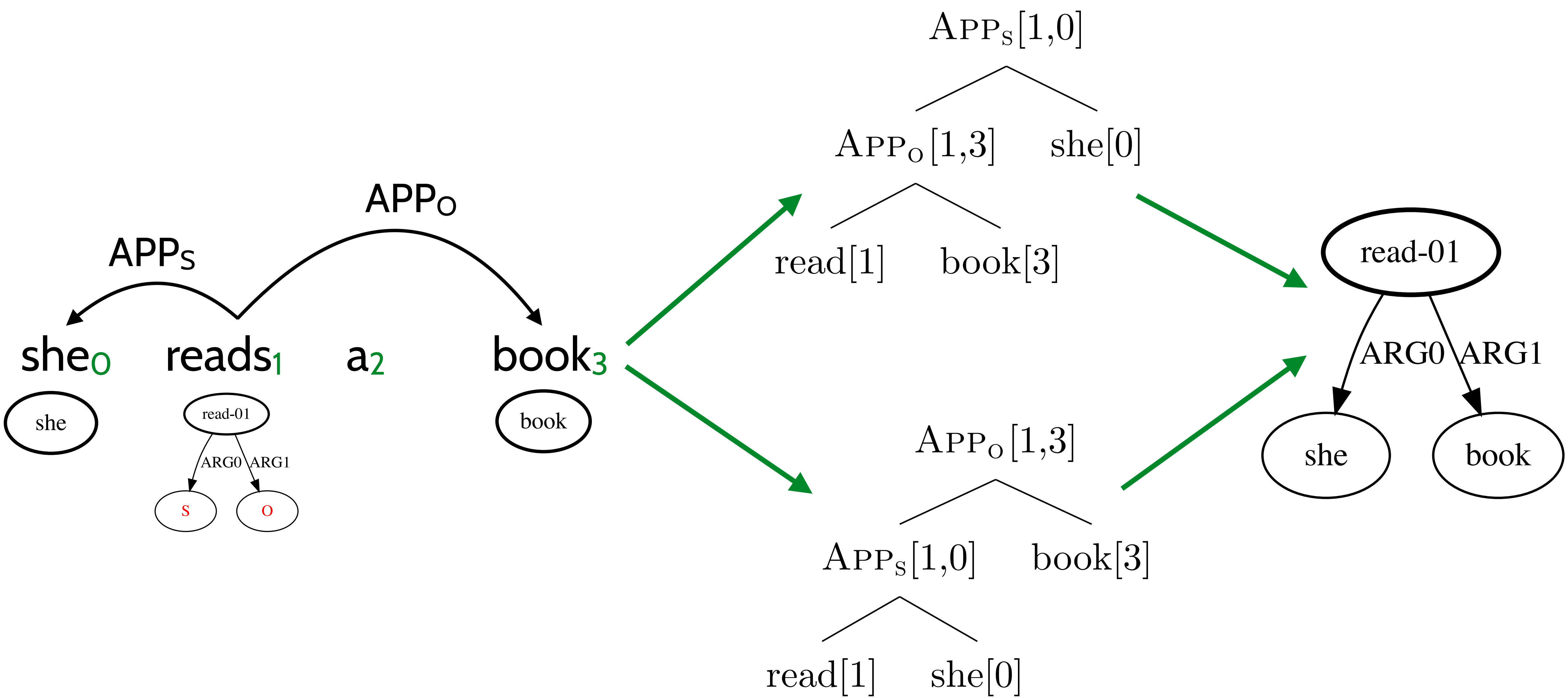


- add 'IGNORE' edges for words that are not represented in semantics. (won't use these in this talk)

AM dependency trees

Crucial: an AM dependency tree defines an AM term only up to reordering within maximal projections, **but** all those terms evaluate to the same AMR!

In other words: **An AM dependency tree underspecifies the AM term, but not the AMR.**



AM dependency trees

- We call an AM dependency tree **well-typed** if there is at least one corresponding well-typed AM term
- Then: every well-typed AM dependency tree produces a unique AMR.

2. In Practice

In Practice

- Task: generate AMRs from sentences
- Idea: train model to predict AM dependency trees
- Can use methods from plain dependency parsing

The task in detail

- **Decoding:** find well-typed AM dependency tree t that maximizes

$$\omega(t) = \sum_{1 \leq i \leq n} \omega(G[i]) + \sum_{o[i,k] \in E(t)} \omega(o[i,k])$$

- **Training:** train a scoring model ω, using the AMR Bank

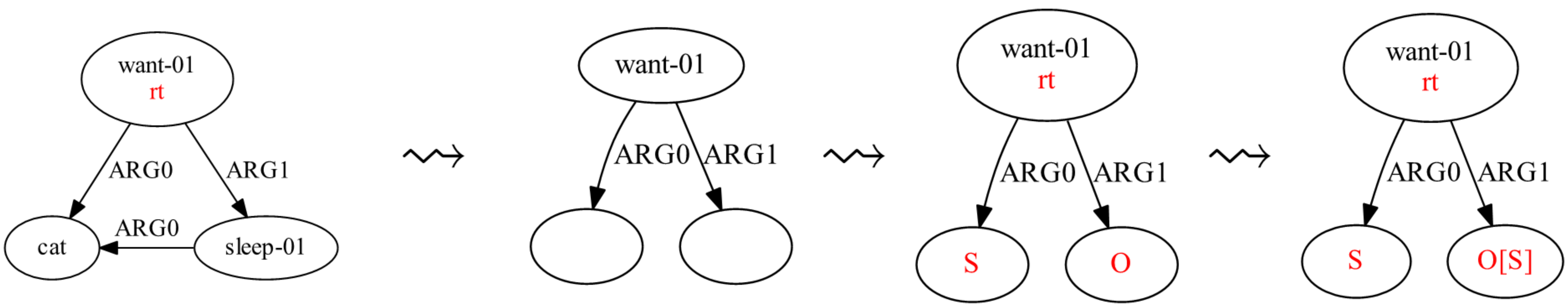
Training data

- AMR Bank contains only sentence-AMR pairs, but we need AM dependency trees to train our model.

Graph decomposition

Get training data for dependency parser:

Step 1: Extract constants, with sources and annotations. Uses graph structure and heuristic alignments.

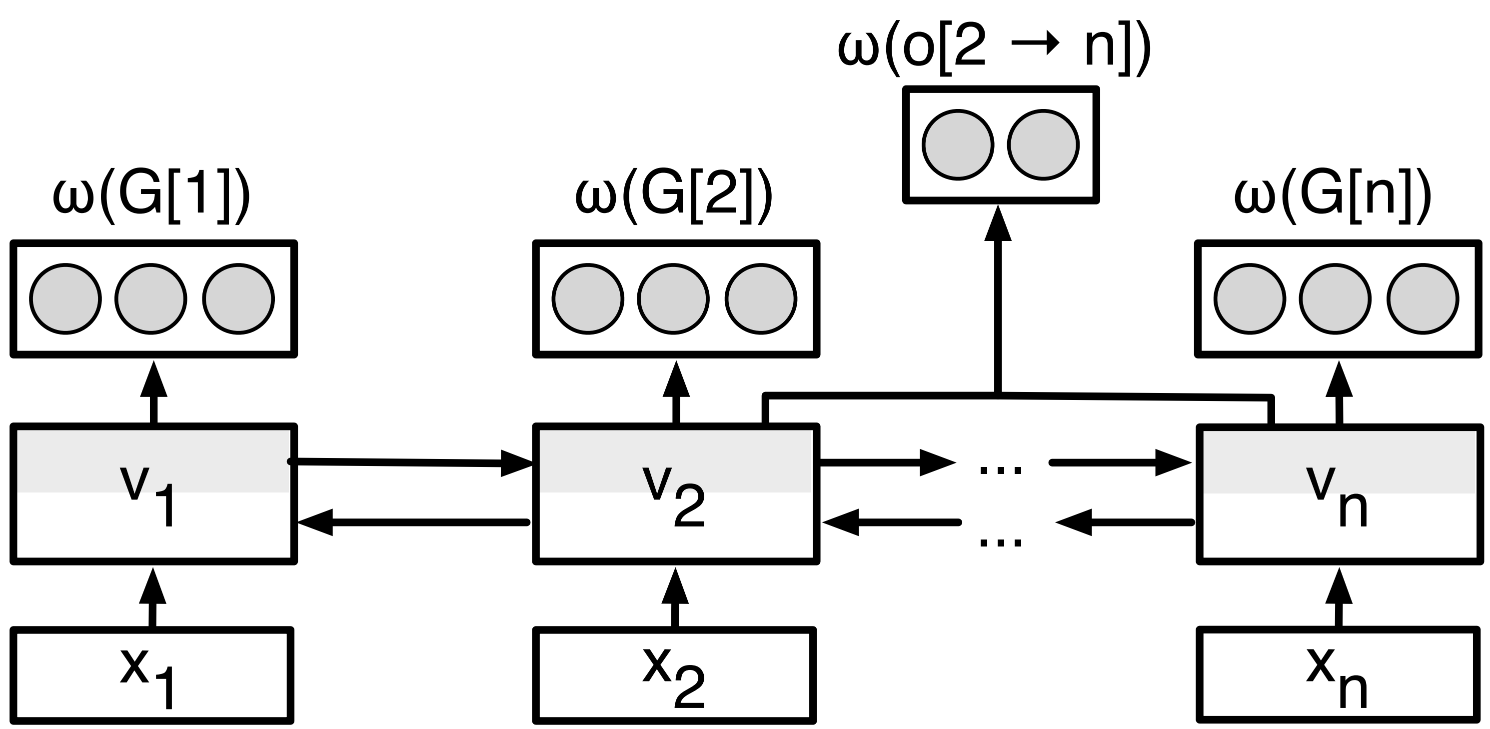


We do not look at the string at this time, and choose source names heuristically.

Step 2: Build AM dependency tree from these constants + alignments.

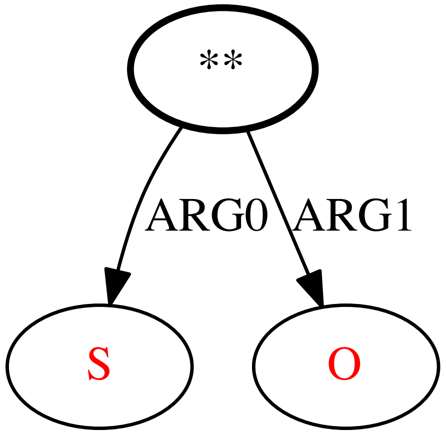
Training

- We follow the general idea of Kiperwasser & Goldberg (2016)
- encode sentence with BiLSTM -> vector v_i for each index i
- predict elementary graph G[i] (or its absence) from v_i
- predict edge o[i,j] from concatenation $v_i \circ v_j$



Training

- predict delexicalized templates for elementary graphs G[i] separately from their labels.



- Template vocabulary size ~2000 (most very rare)

- Tagger accuracy: 73% (correct template in top 5: ~90%)

Decoding

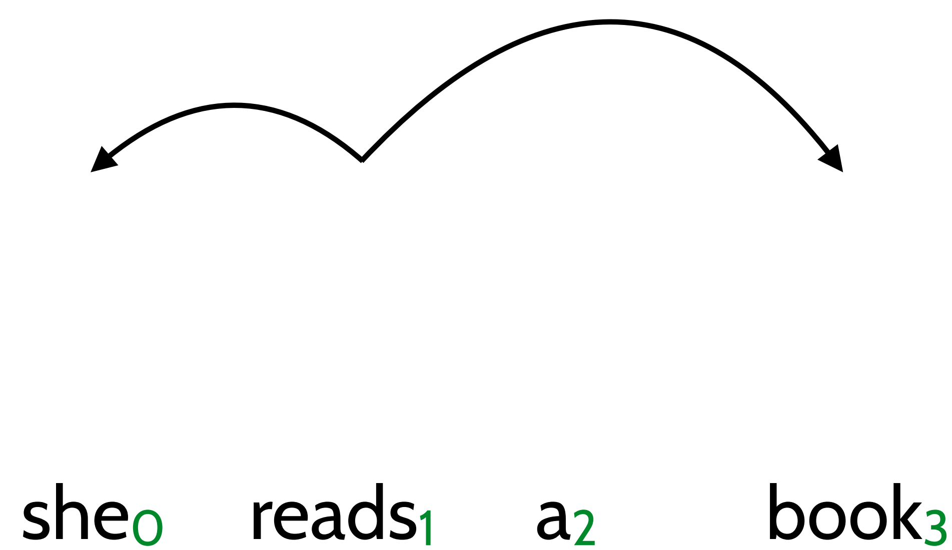
- **Decoding:** find well-typed AM dependency tree t that maximizes

$$\omega(t) = \sum_{1 \leq i \leq n} \omega(G[i]) + \sum_{o[i,k] \in E(t)} \omega(o[i,k])$$

Decoding

Option 1: Fixed tree decoder

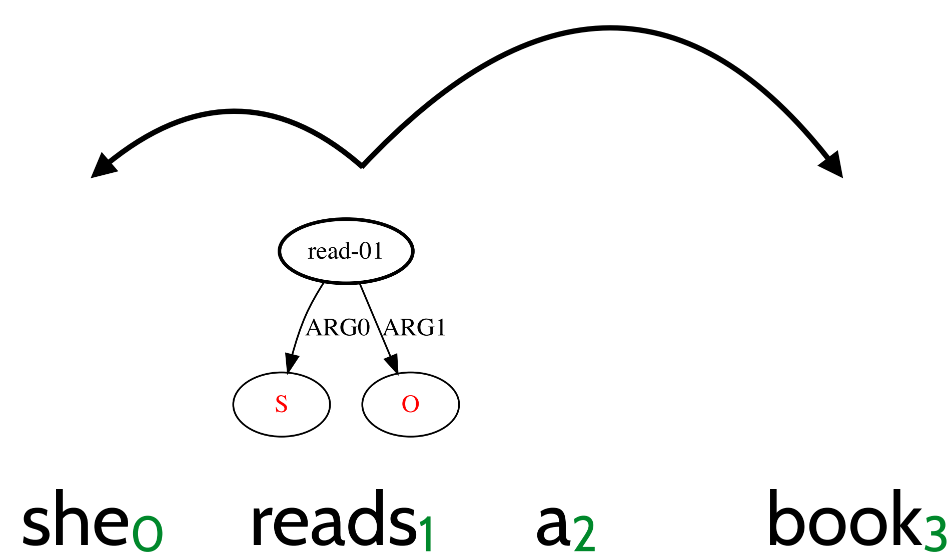
- First, predict an unlabeled dependency tree using standard methods.
- Second, find the best well-typed combination of elementary graphs G[i] and operations o[i,j] using a viterbi-style algorithm.
- Can produce non-projective dependency trees.
- Without type-checking, over 70% of analyses are not well-typed and fail.



Decoding

Option 1: Fixed tree decoder

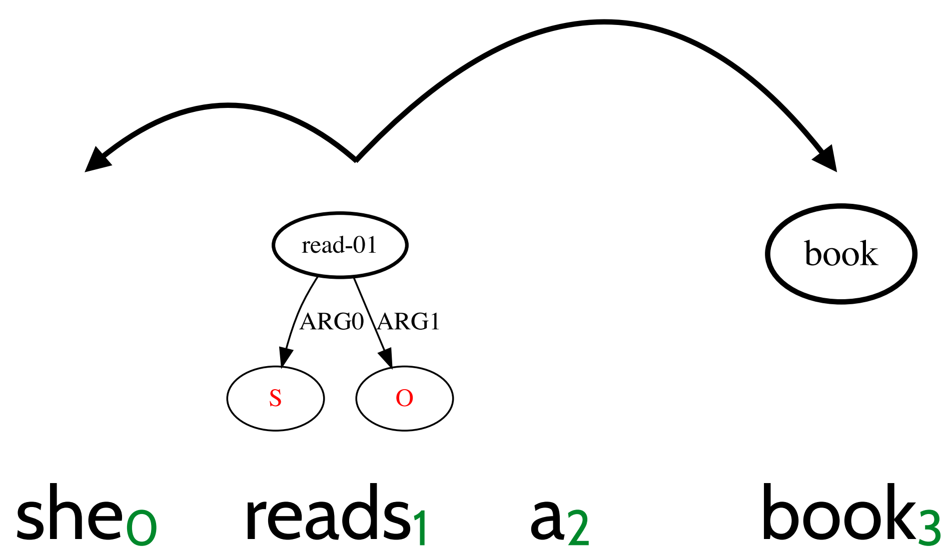
- First, predict an unlabeled dependency tree using standard methods.
- Second, find the best well-typed combination of elementary graphs G[i] and operations o[i,j] using a viterbi-style algorithm.
- Can produce non-projective dependency trees.
- Without type-checking, over 70% of analyses are not well-typed and fail.



Decoding

Option 1: Fixed tree decoder

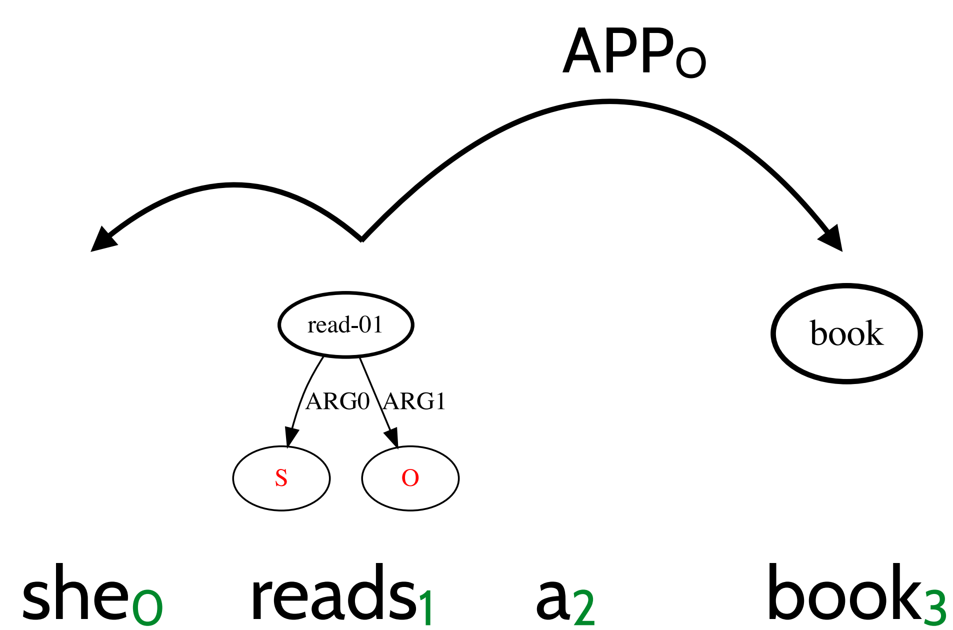
- First, predict an unlabeled dependency tree using standard methods.
- Second, find the best well-typed combination of elementary graphs G[i] and operations o[i,j] using a viterbi-style algorithm.
- Can produce non-projective dependency trees.
- Without type-checking, over 70% of analyses are not well-typed and fail.



Decoding

Option 1: Fixed tree decoder

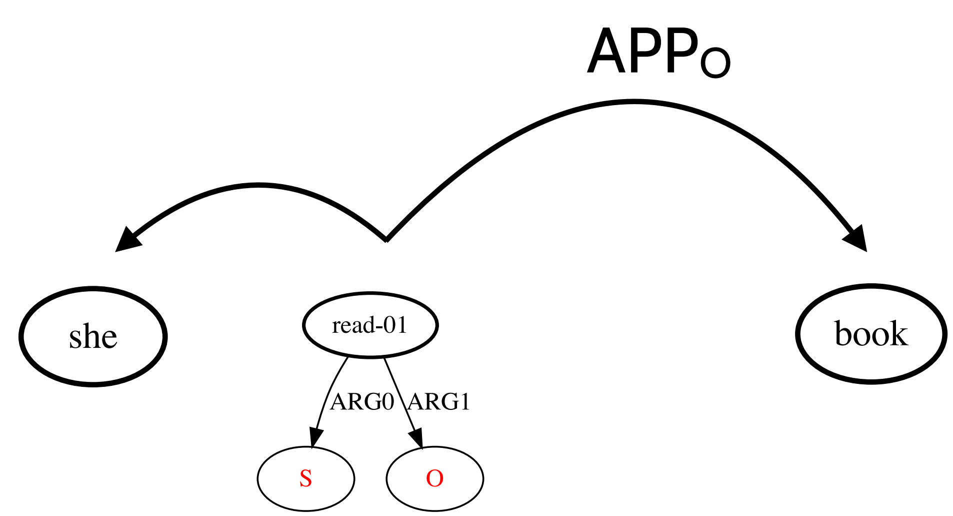
- First, predict an unlabeled dependency tree using standard methods.
- Second, find the best well-typed combination of elementary graphs G[i] and operations o[i,j] using a viterbi-style algorithm.
- Can produce non-projective dependency trees.
- Without type-checking, over 70% of analyses are not well-typed and fail.



Decoding

Option 1: Fixed tree decoder

- First, predict an unlabeled dependency tree using standard methods.
- Second, find the best well-typed combination of elementary graphs G[i] and operations o[i,j] using a viterbi-style algorithm.
- Can produce non-projective dependency trees.
- Without type-checking, over 70% of analyses are not well-typed and fail.

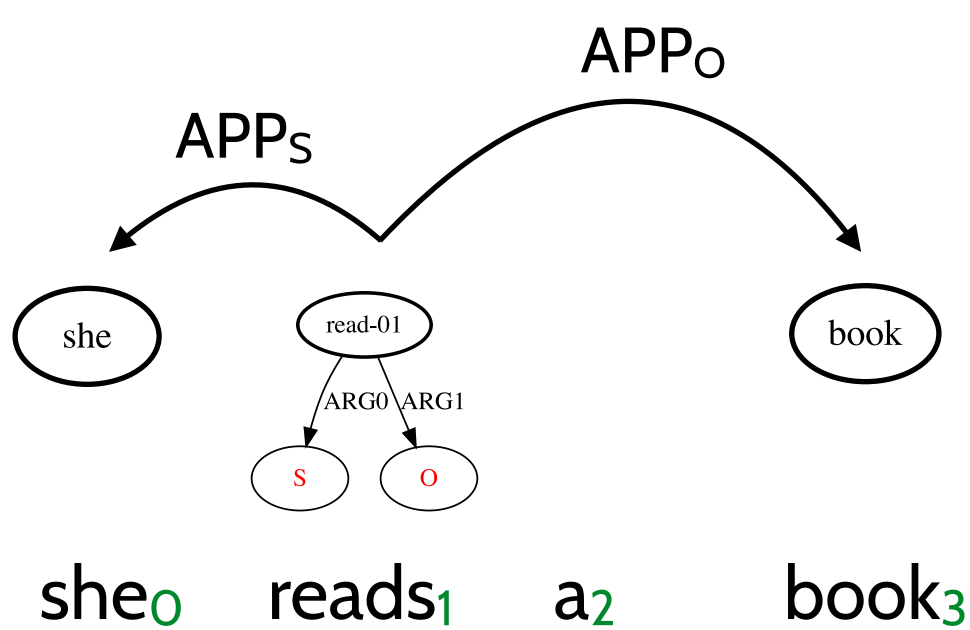


she_0 reads_1 a_2 book_3

Decoding

Option 1: Fixed tree decoder

- First, predict an unlabeled dependency tree using standard methods.
- Second, find the best well-typed combination of elementary graphs G[i] and operations o[i,j] using a viterbi-style algorithm.
- Can produce non-projective dependency trees.
- Without type-checking, over 70% of analyses are not well-typed and fail.



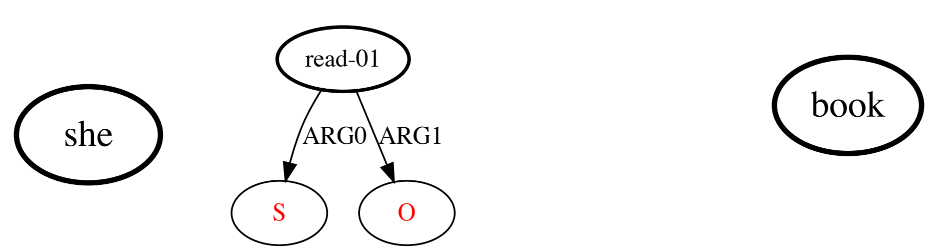
Decoding

Option 2: projective decoder

- combine adjacent spans and their partial results -> CKY-like parser.
- Consequence: Decoder builds its own tree to fit type constraints, but has strong projectivity constraints.

[ ] [S,O] [ ]

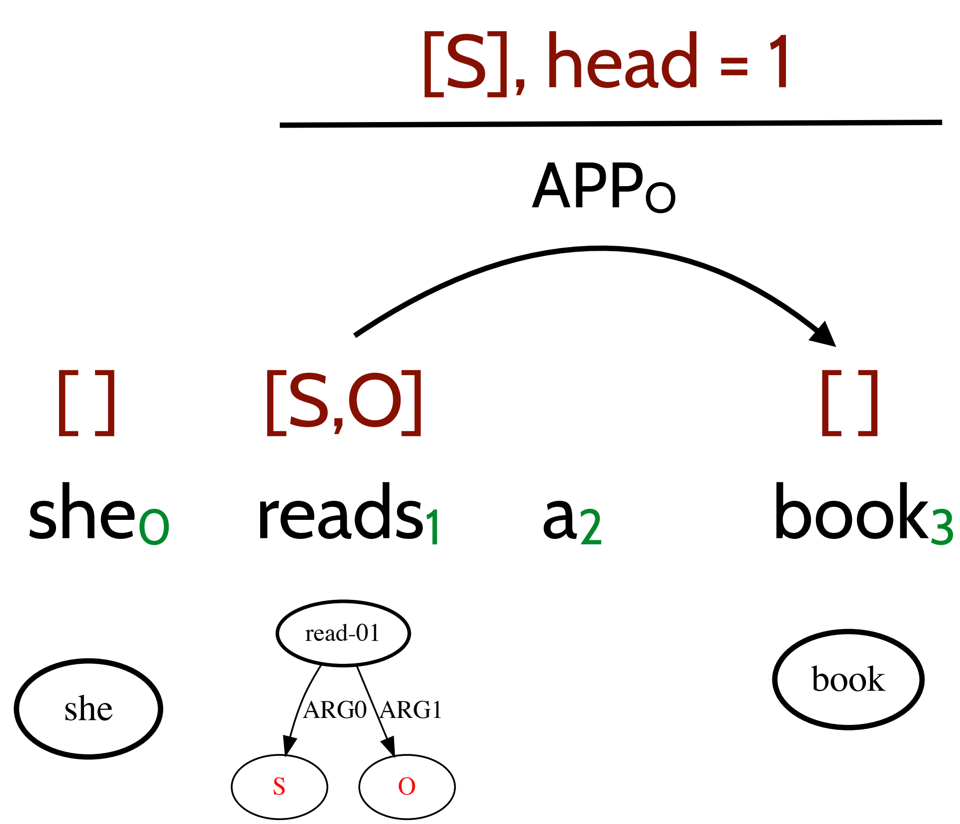
she$_0$ reads$_1$ a$_2$ book$_3$



Decoding

Option 2: projective decoder

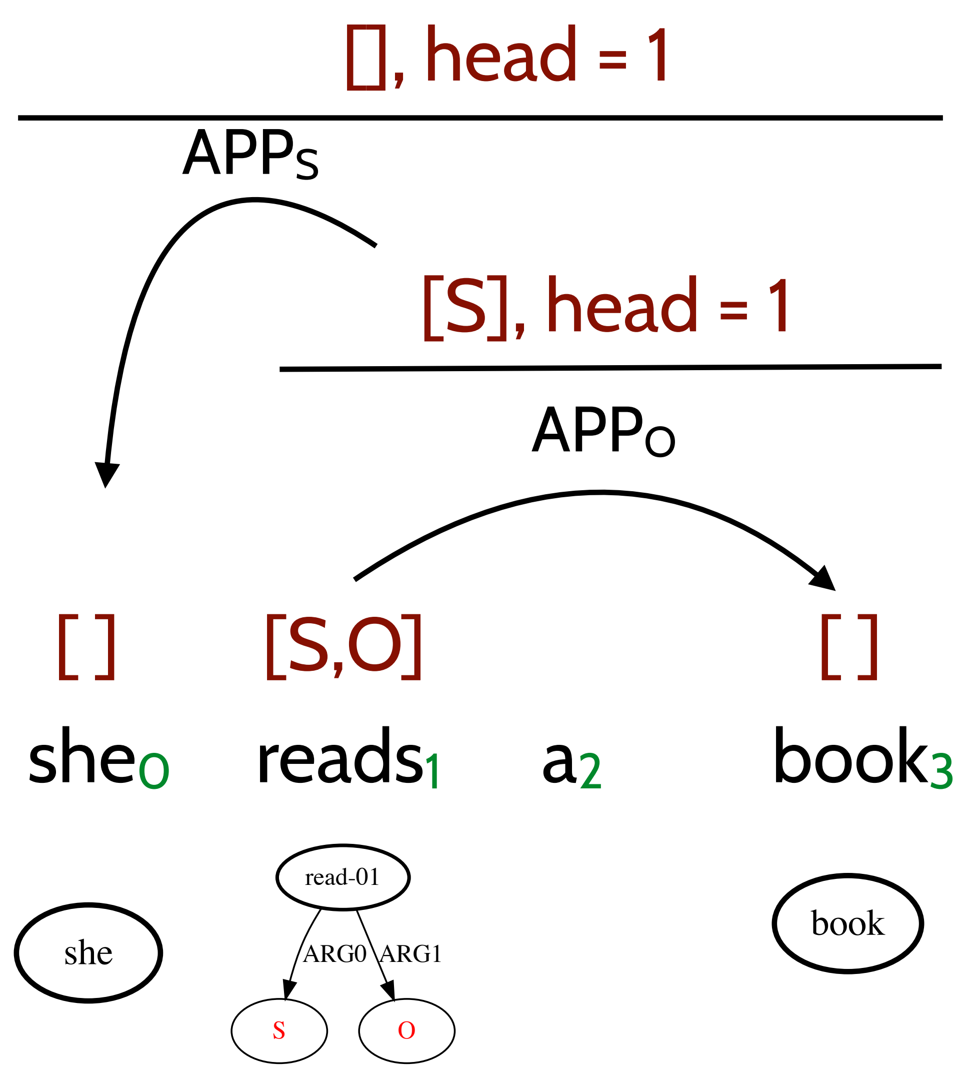
- combine adjacent spans and their partial results -> CKY-like parser.
- Consequence: Decoder builds its own tree to fit type constraints, but has strong projectivity constraints.



Decoding

Option 2: projective decoder

- combine adjacent spans and their partial results -> CKY-like parser.
- Consequence: Decoder builds its own tree to fit type constraints, but has strong projectivity constraints.



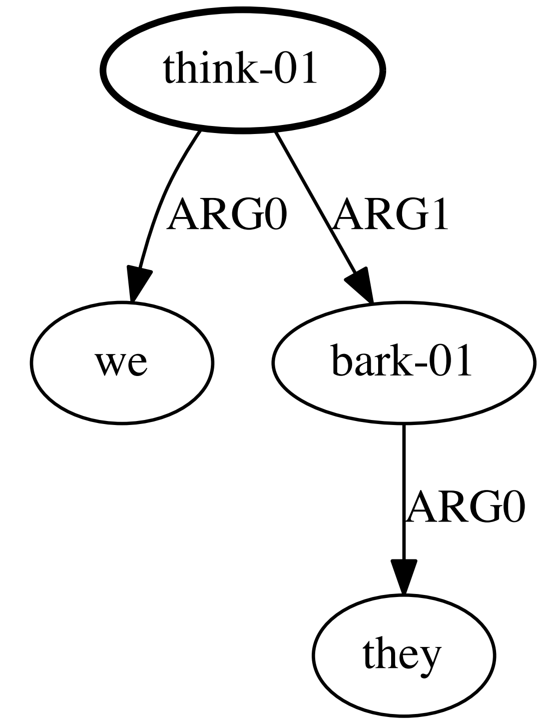
Results

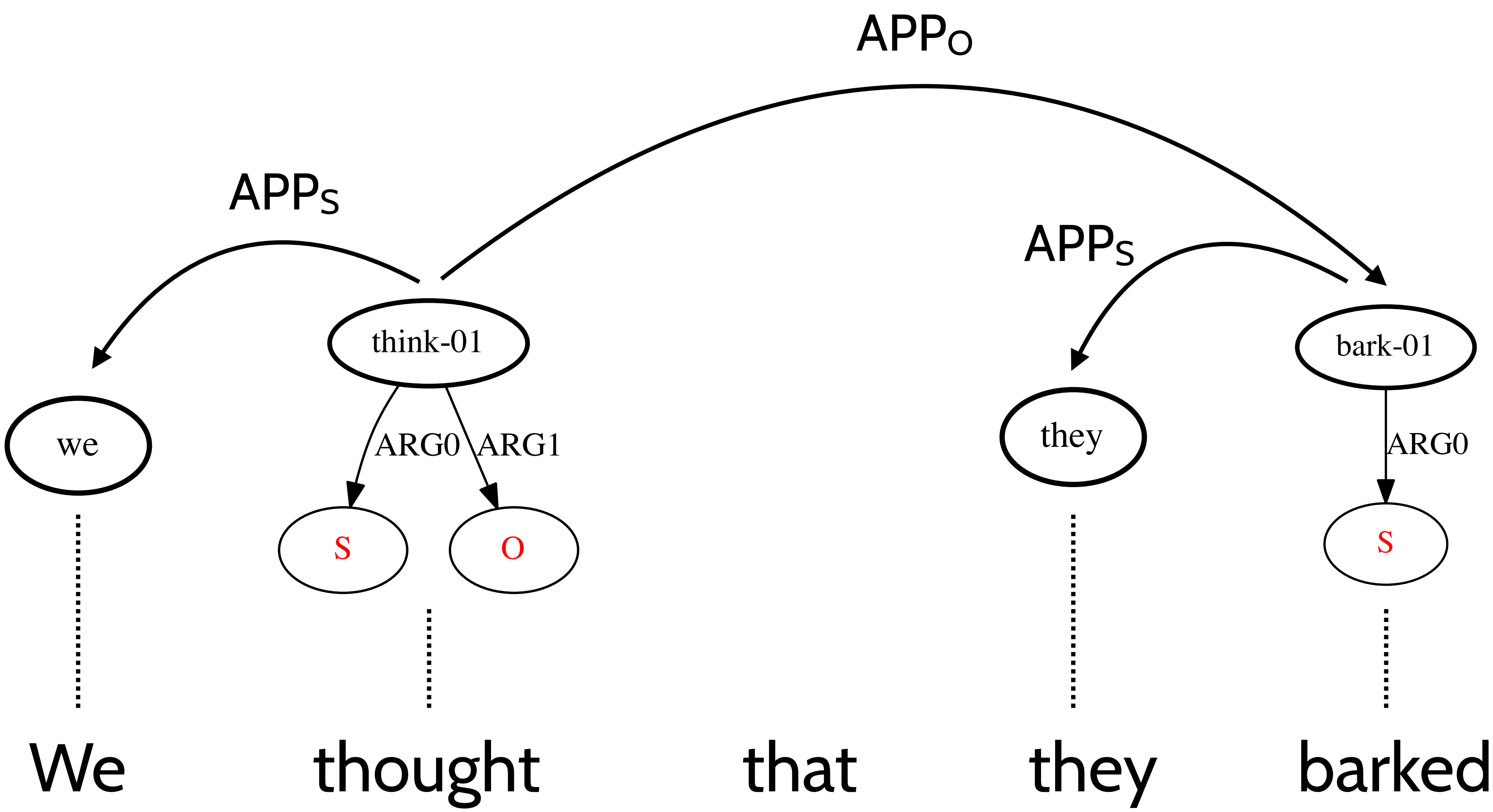
JAMR (Flanigan et al. 2016)	67
Damonte et al. (2017)	64
Foland and Martin (2017)	70.7
Our JAMR-style baseline	65.2
CAMR (Wang et al. 2015)	66.5
van Noord and Bos (2017)	68.5
Our projective decoder	70.1
Our fixed tree decoder	69.1

Results on LDC2015E86 dataset

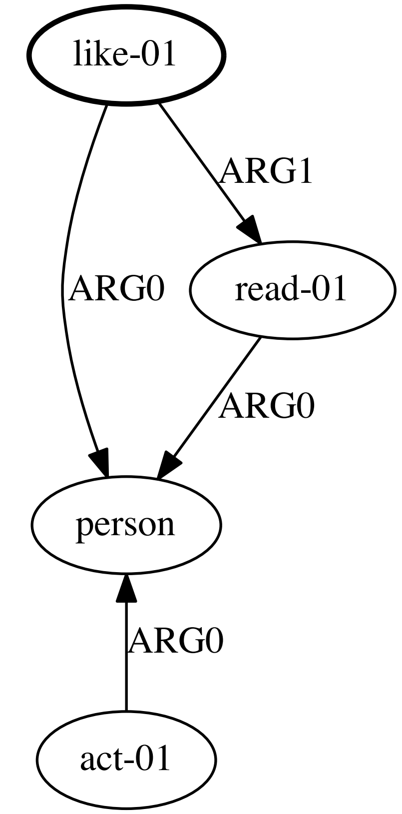
3. Examples

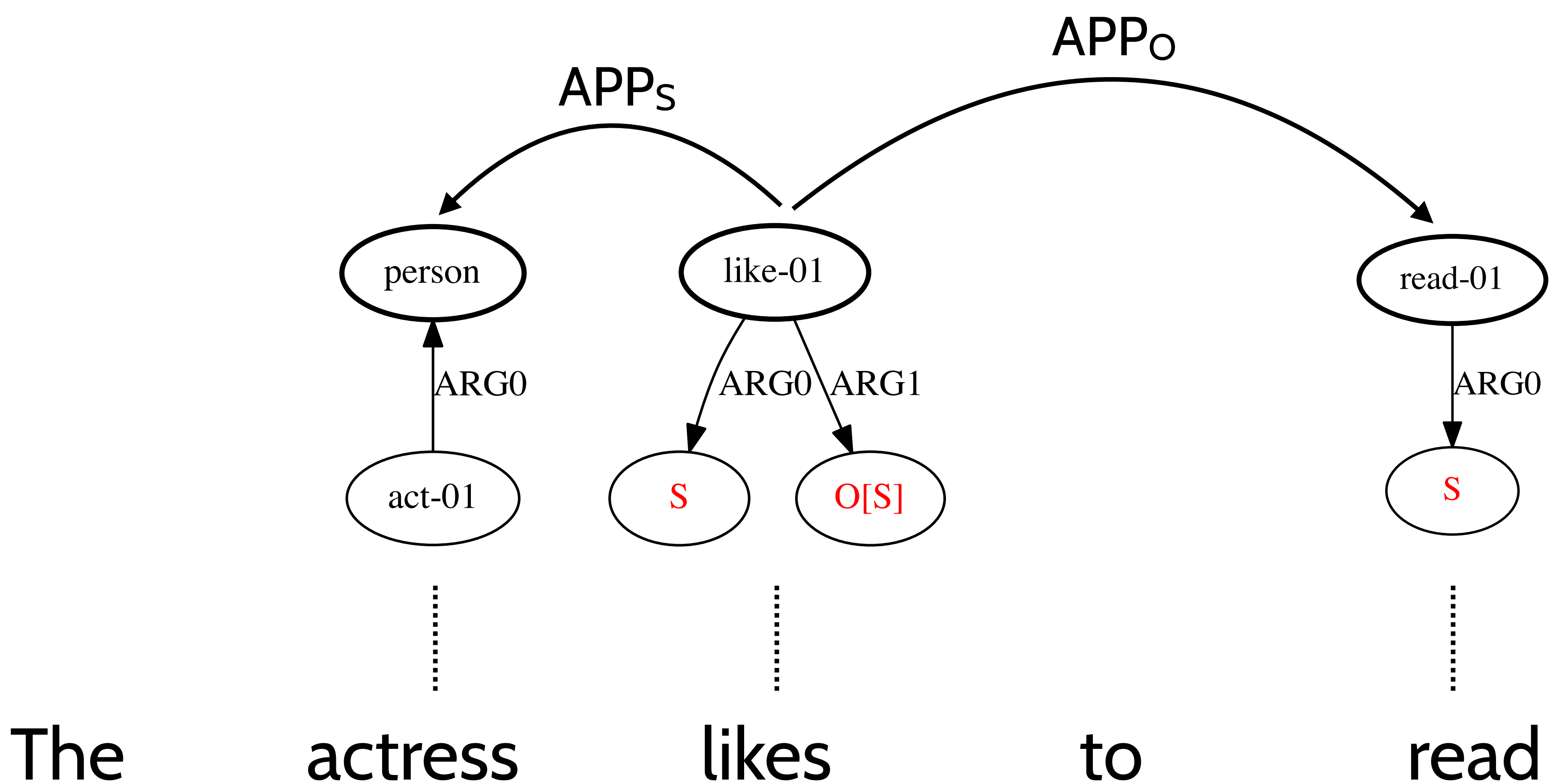
Ex 1: Basic


think-01
ARG0
ARG1
we
bark-01
ARG0
they

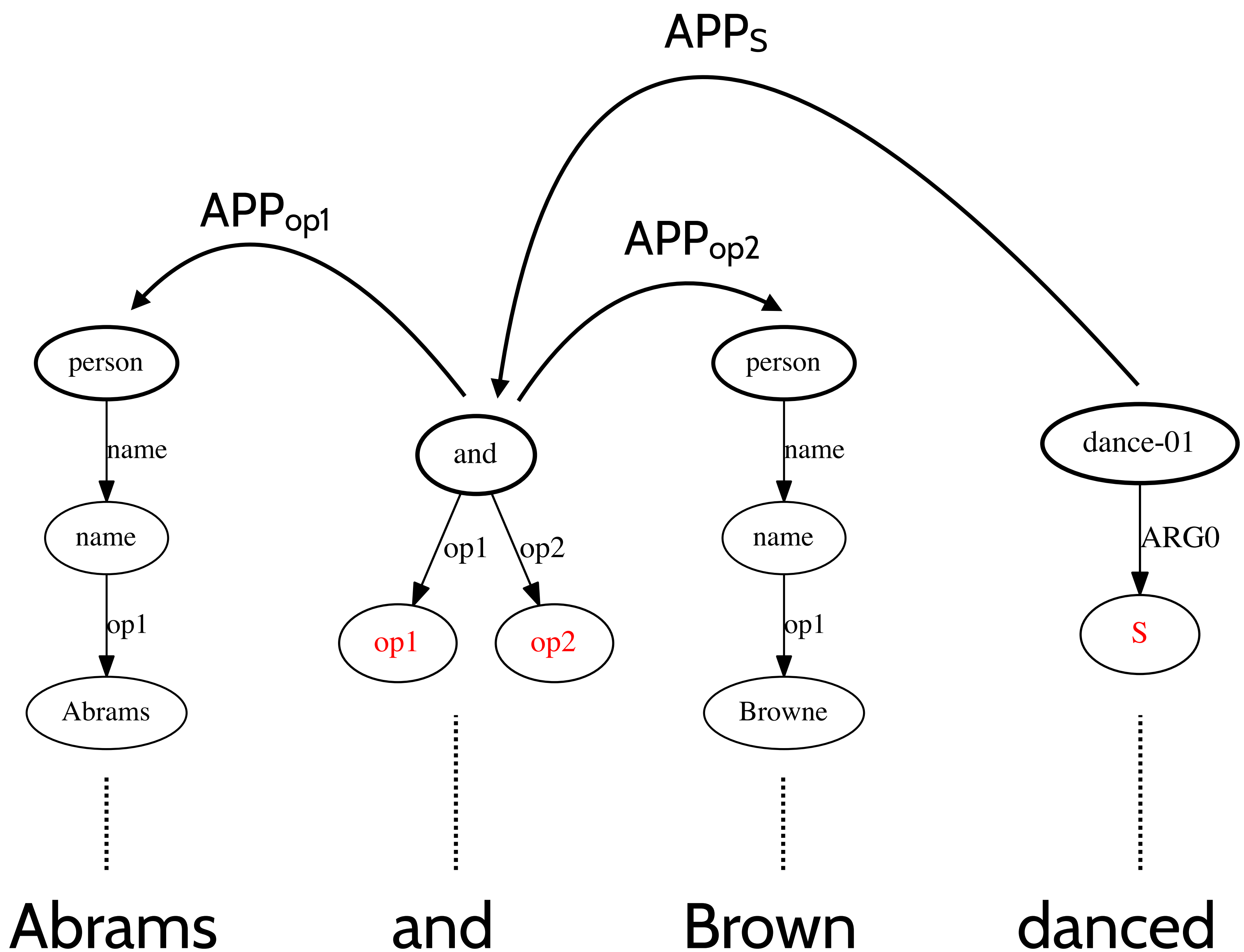

APP$_O$
APP$_S$
APP$_S$
think-01
ARG0
ARG1
S
O
we
they
bark-01
ARG0
S
We
thought
that
they
barked

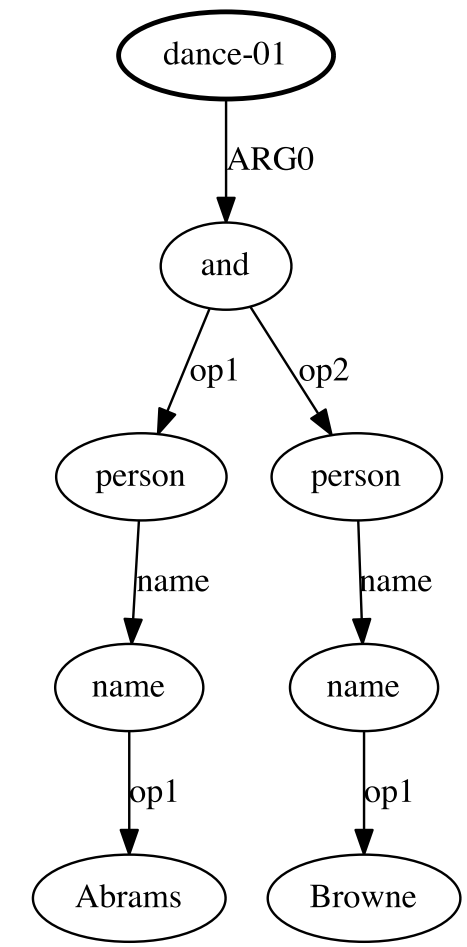
Ex 2: Control


like-01
ARG1
ARG0
read-01
ARG0
person
ARG0
act-01

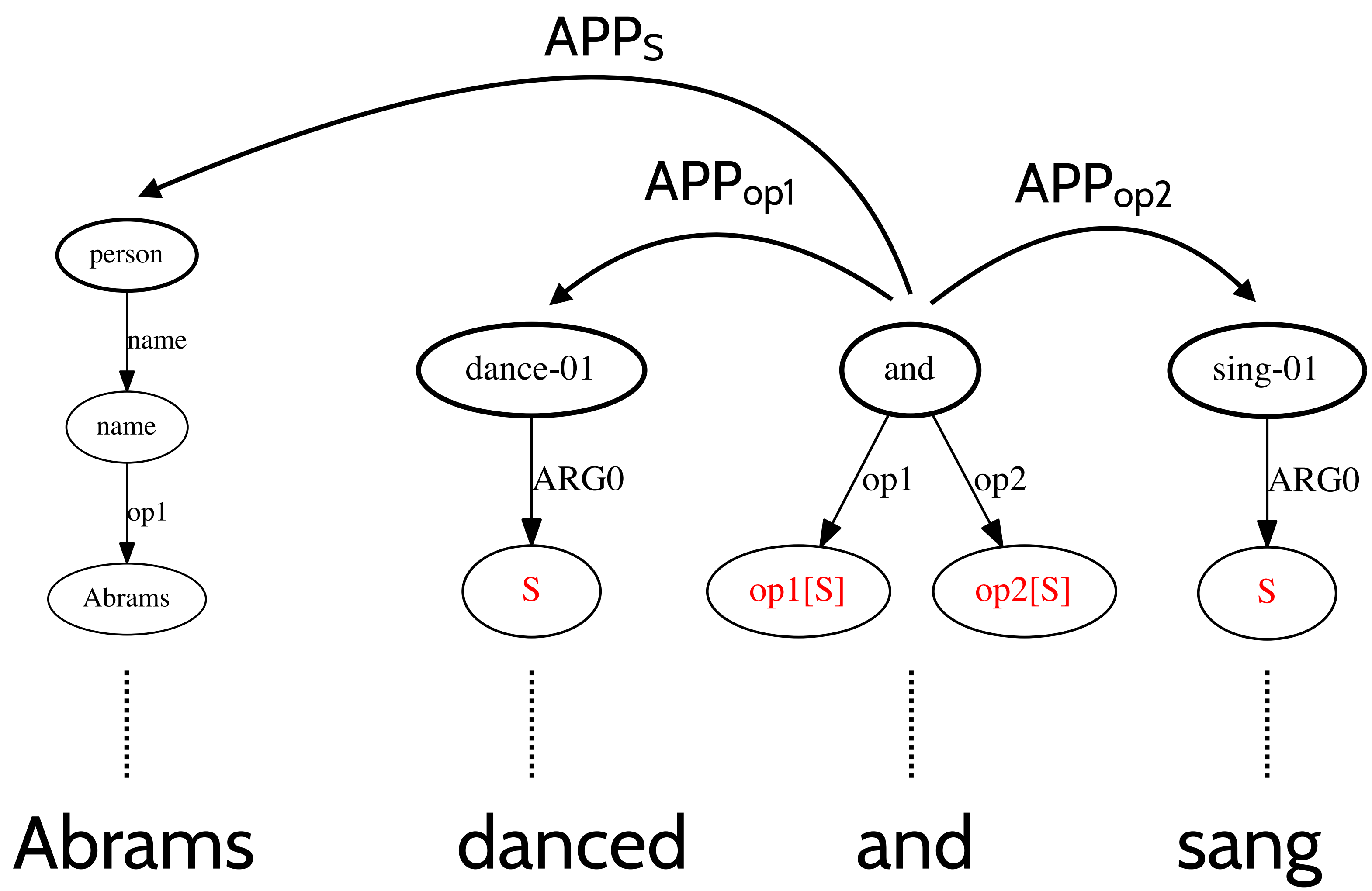

APP_O
APP_S
person
like-01
read-01
ARG0
ARG0
ARG1
ARG0
act-01
S
O[S]
S
The actress likes to read

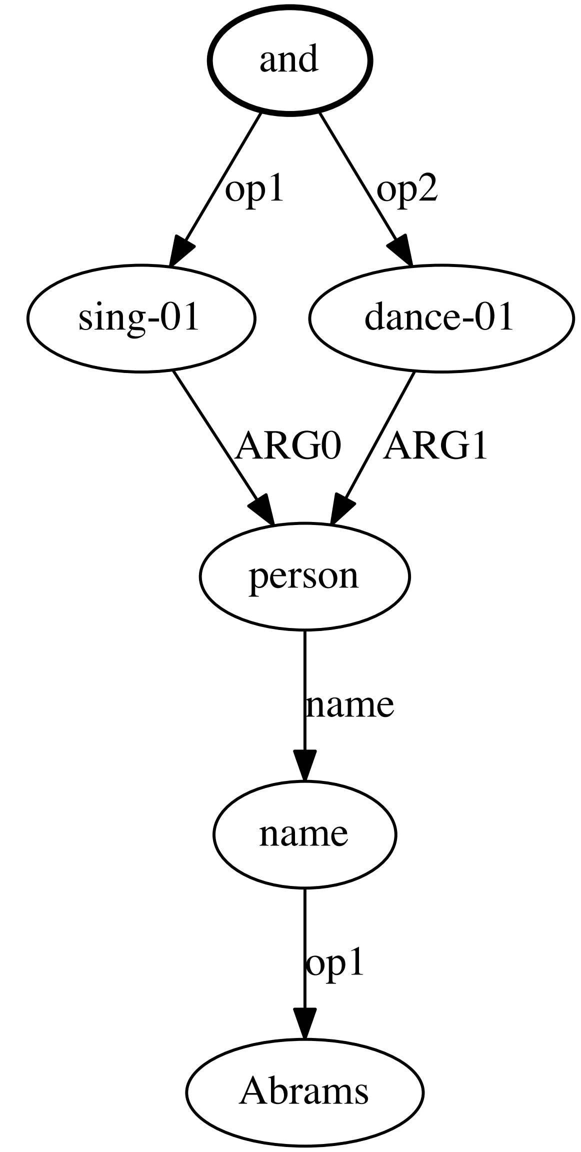
Ex 3: Coordination



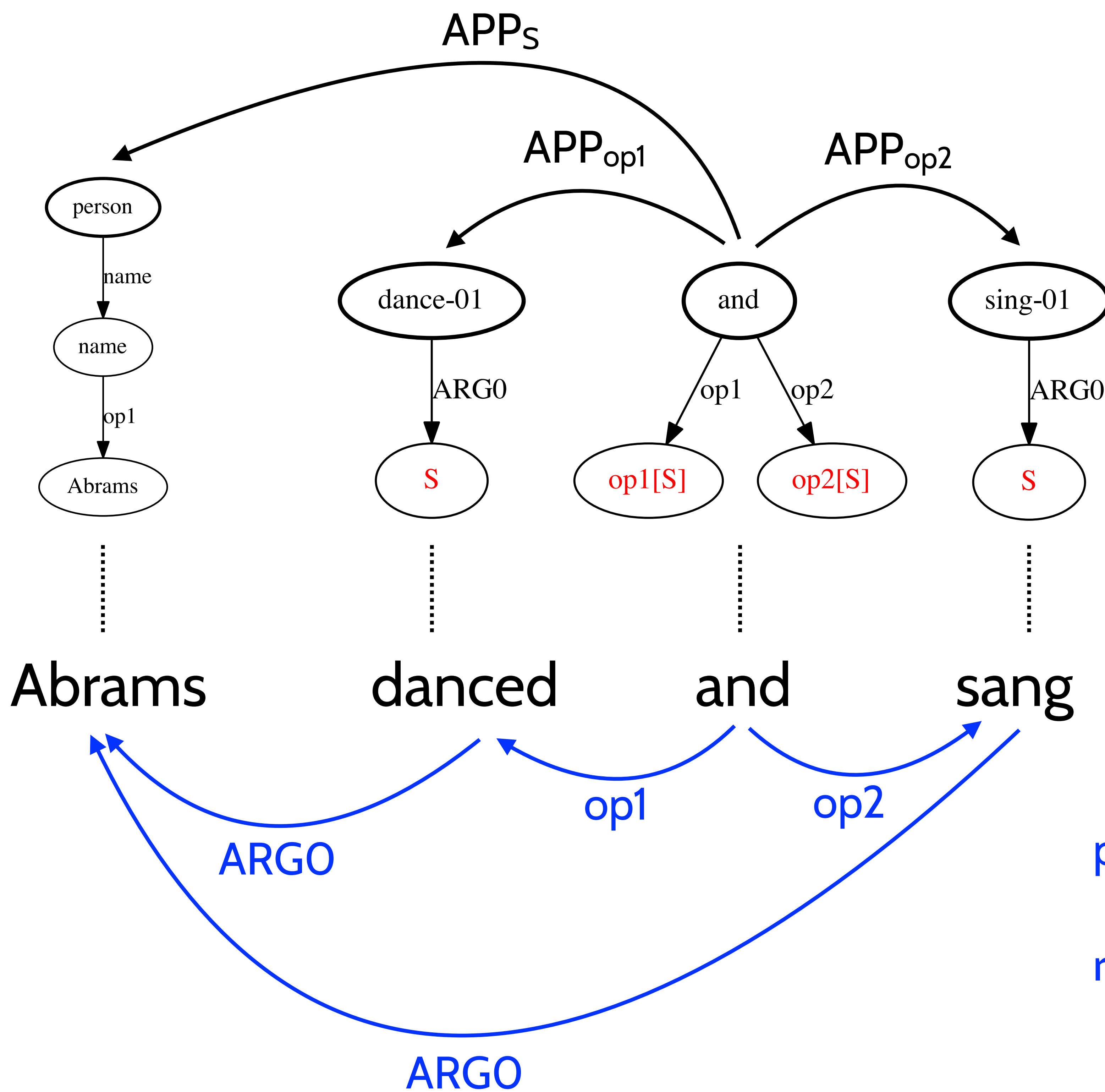


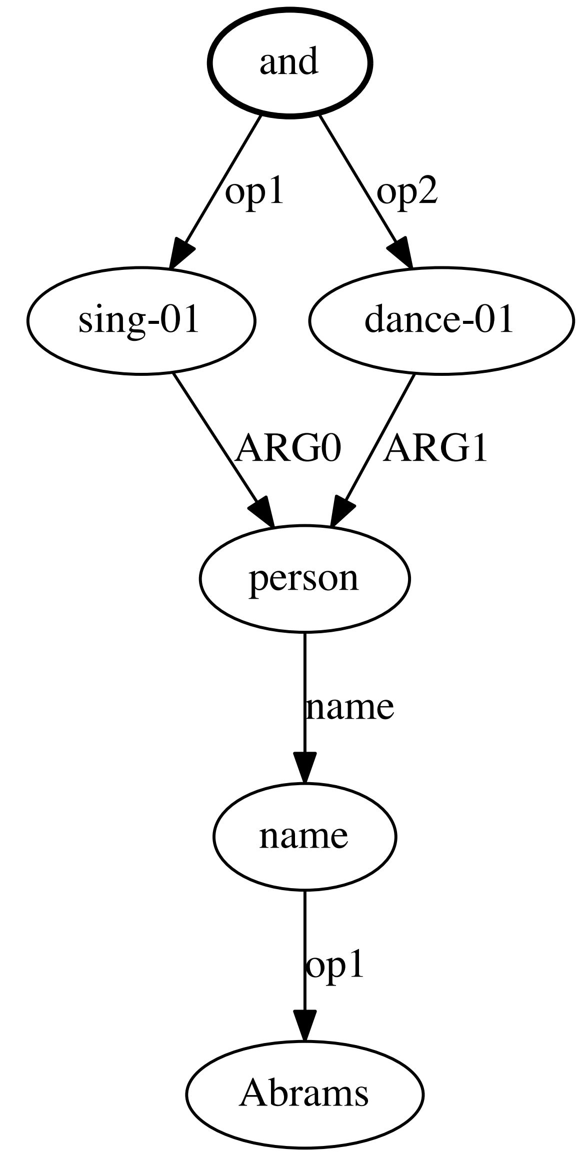
Ex 3b: Some Observations





Ex 3b: Some Observations

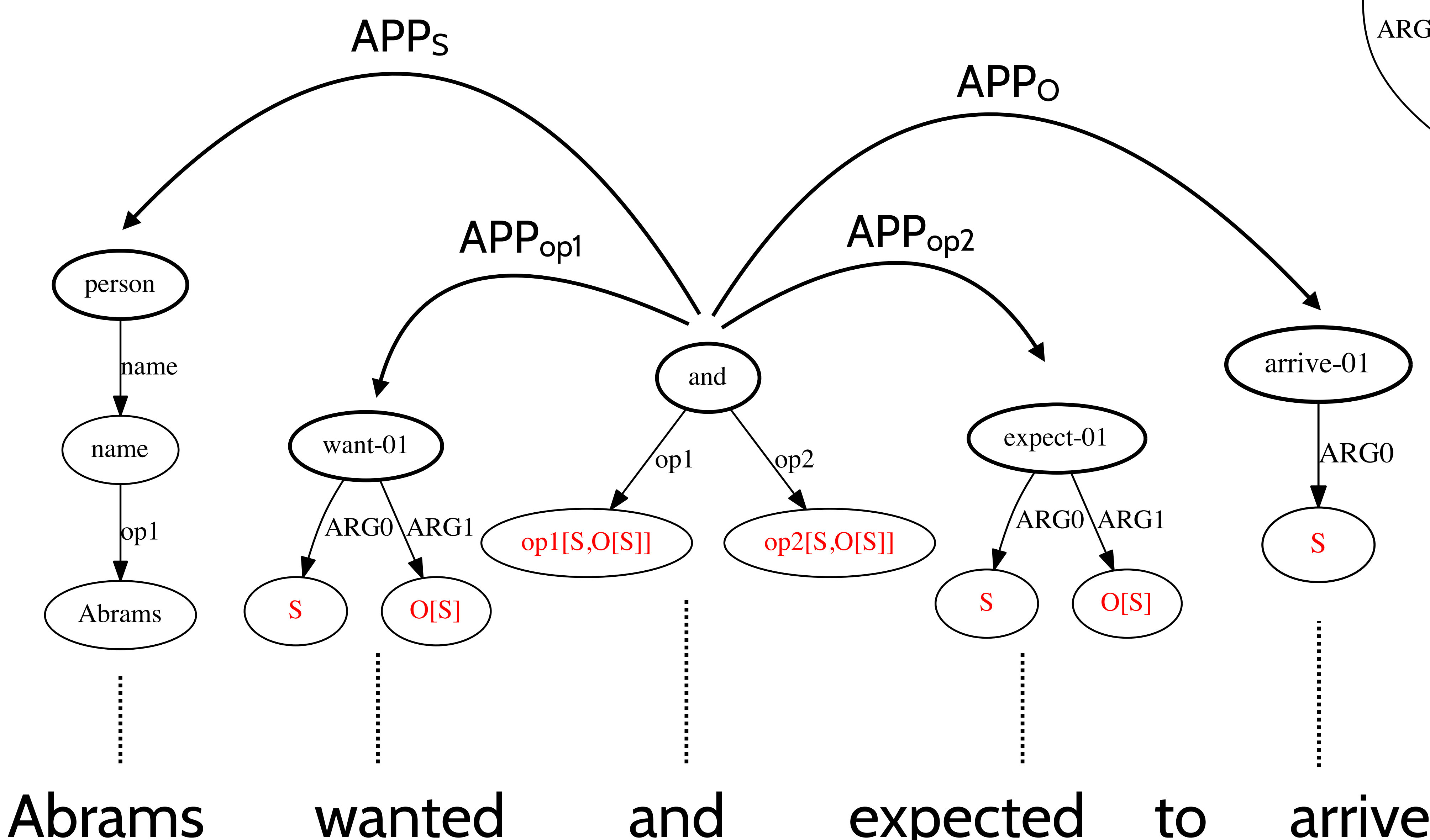


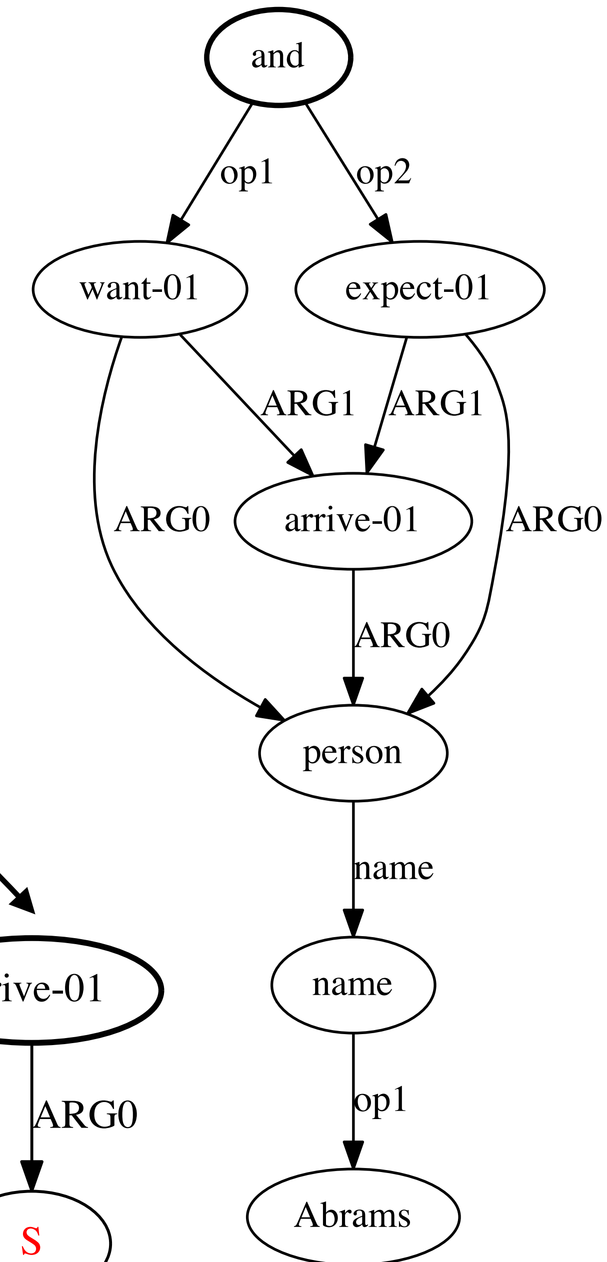


projected AMR edges

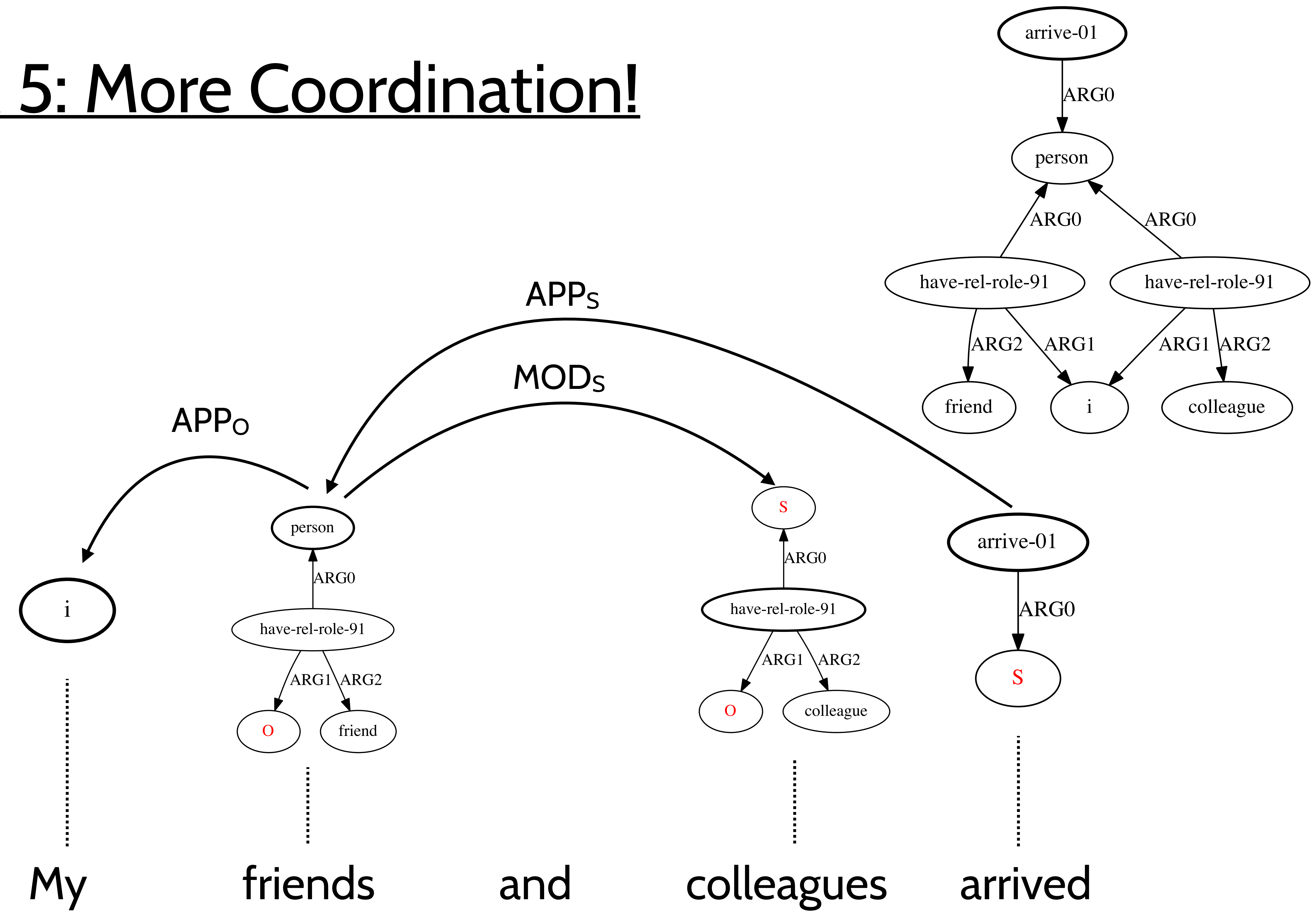
reminiscent of enhanced UD?

Ex 4: Coordination of control verbs

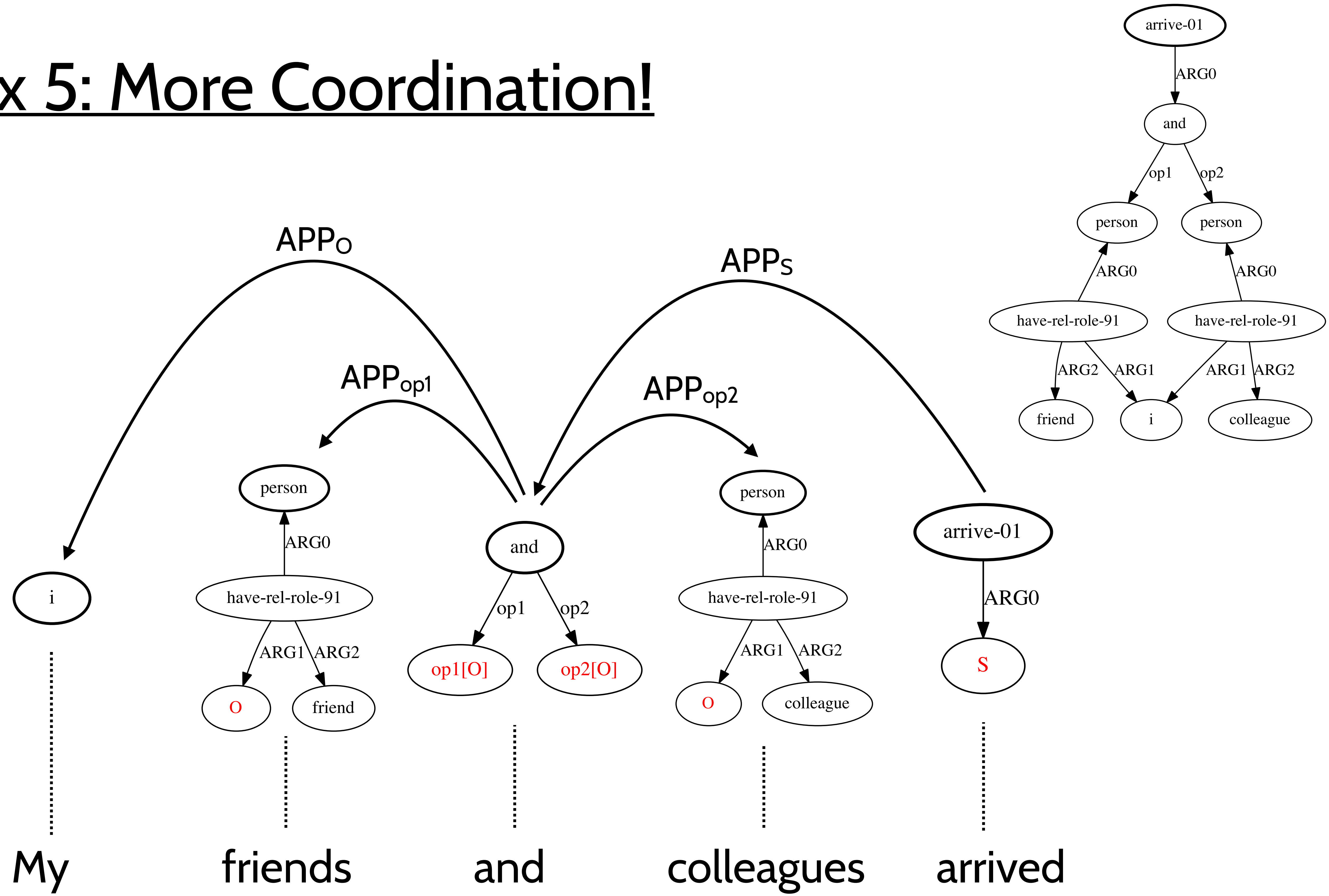




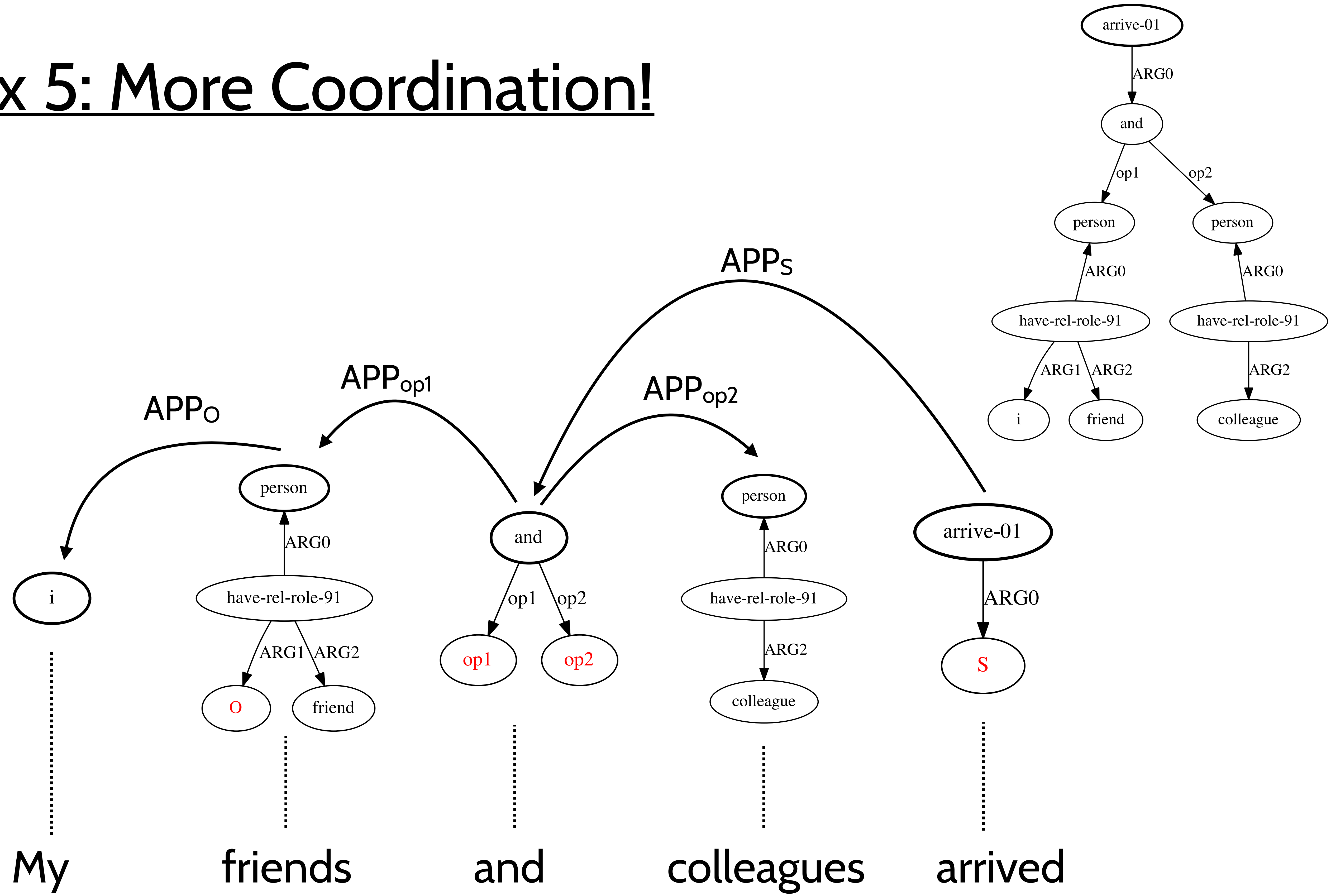
Ex 5: More Coordination!



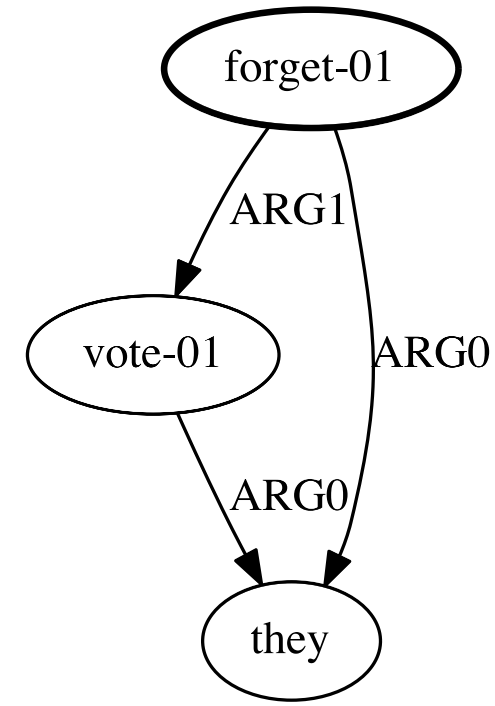
Ex 5: More Coordination!



Ex 5: More Coordination!

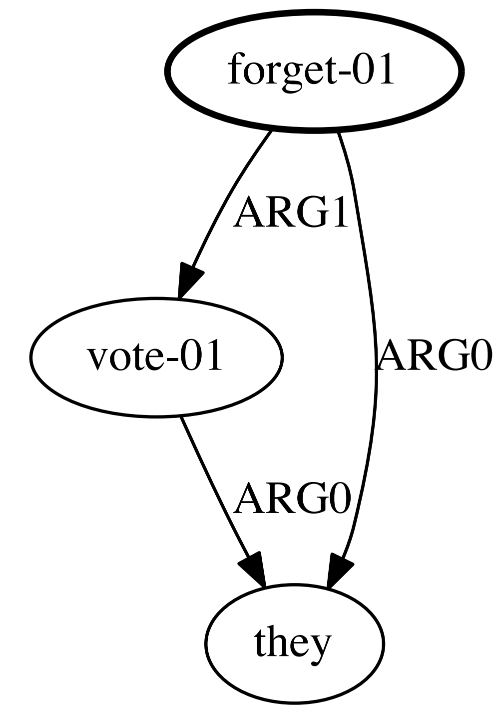


Problem 1: Coreference

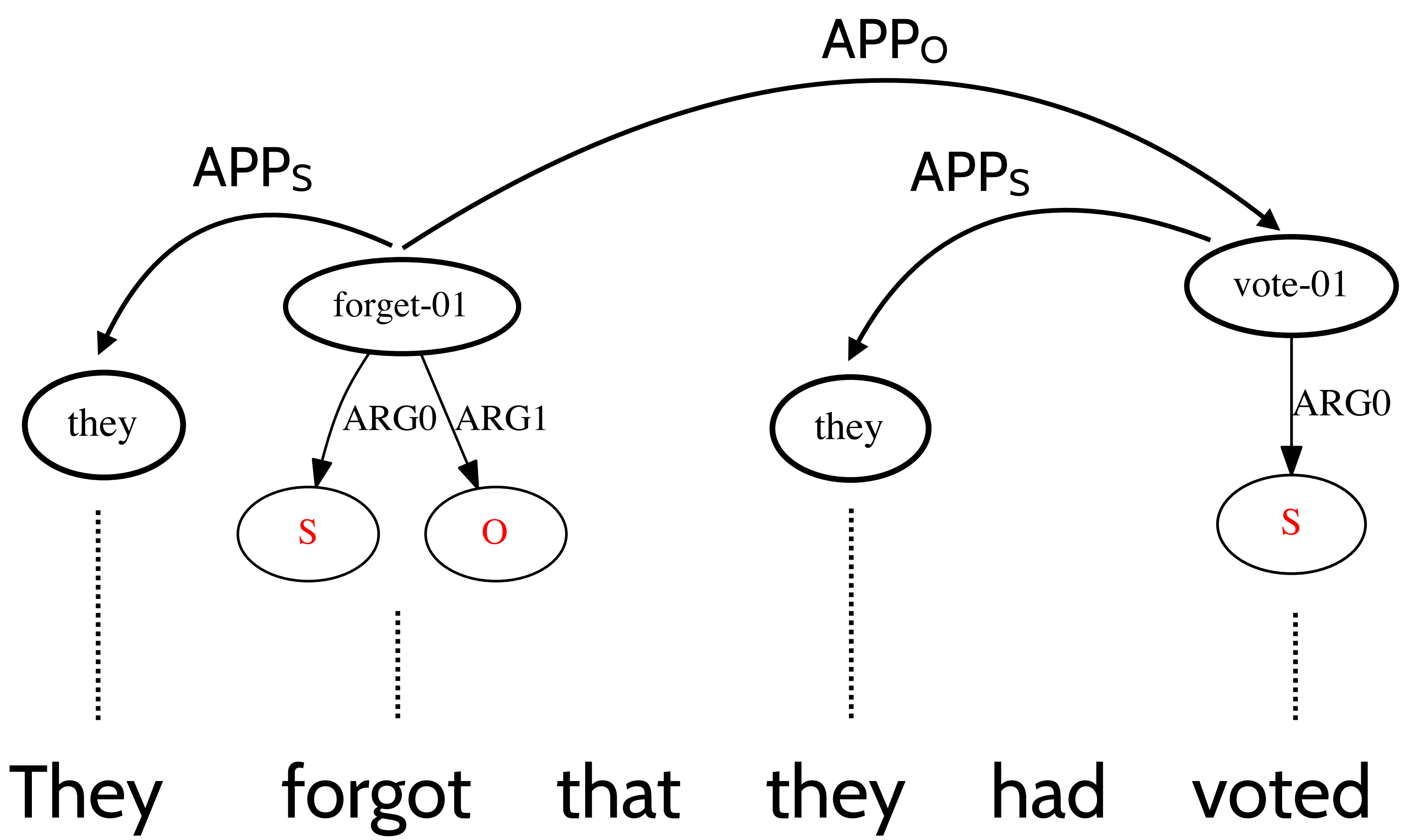


They forgot that they had voted

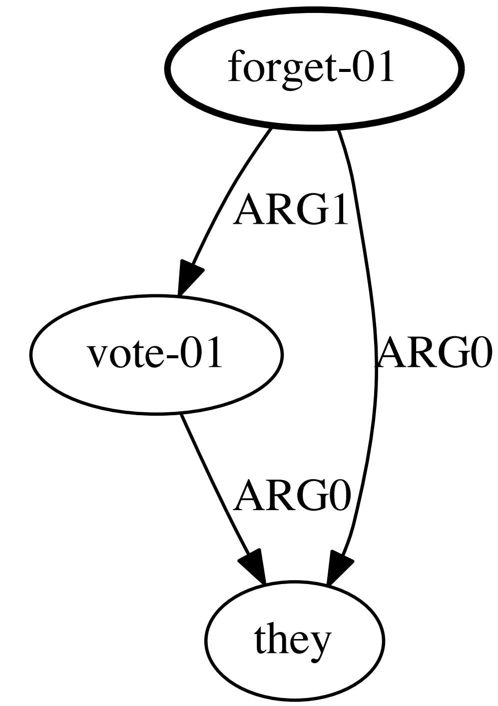
Problem 1: Coreference



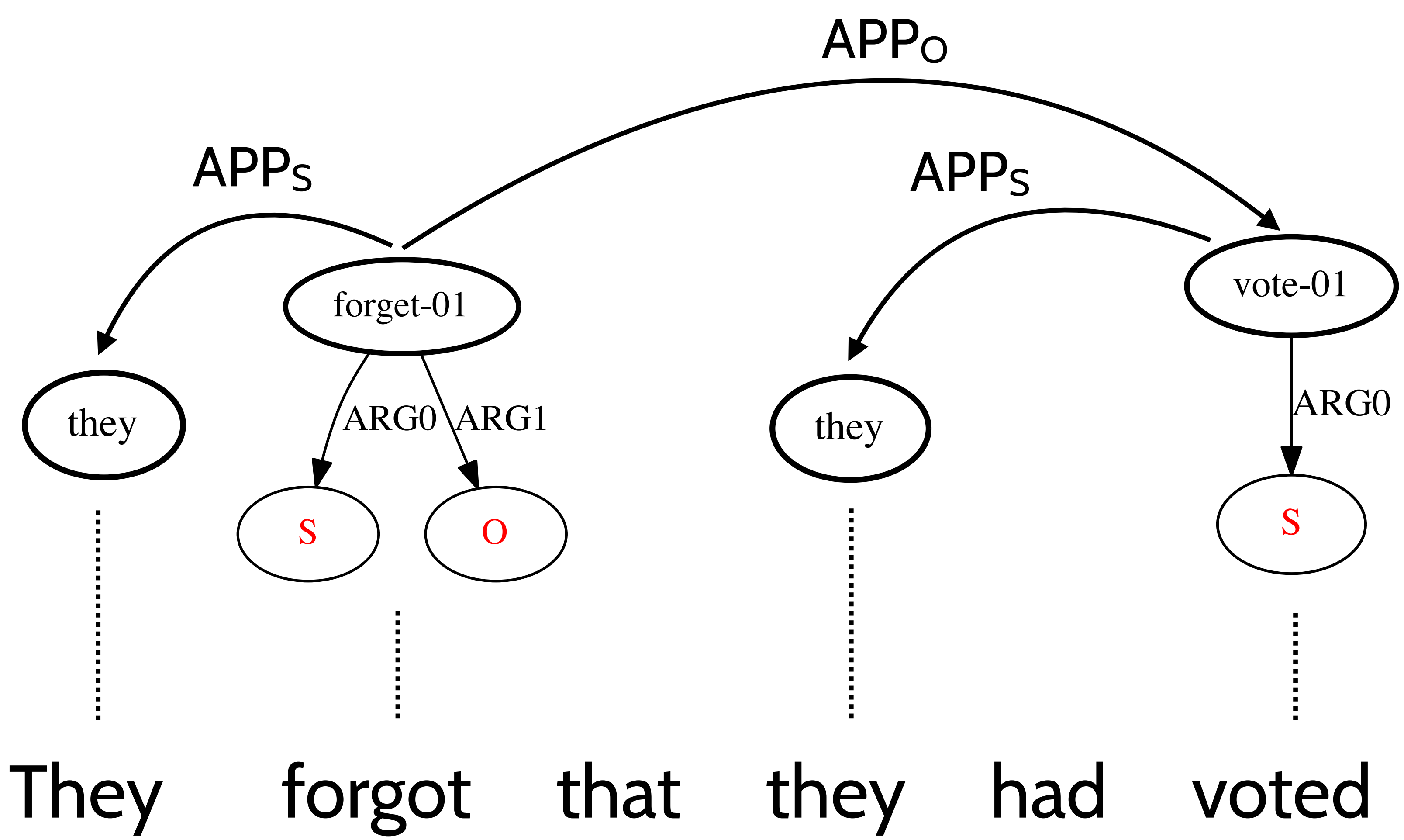
One would think something like this:



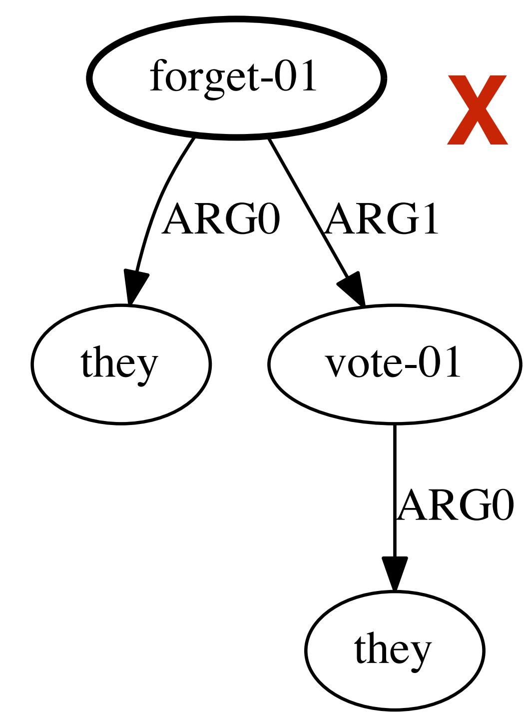
Problem 1: Coreference



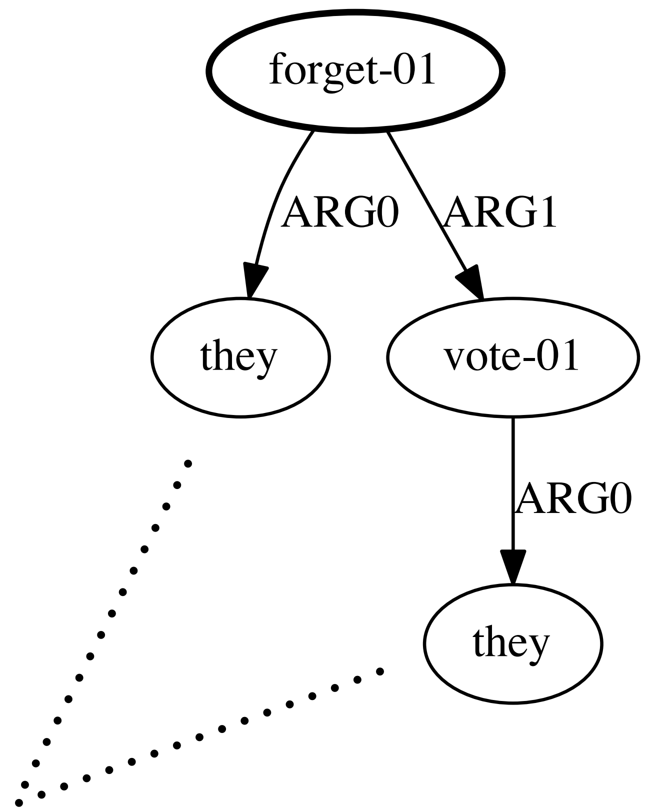
One would think something like this:



But this yields the wrong graph:

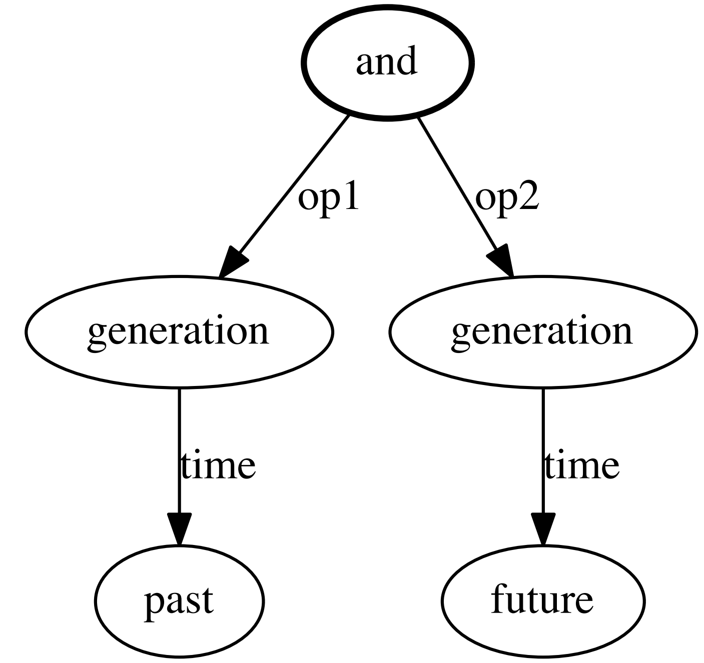


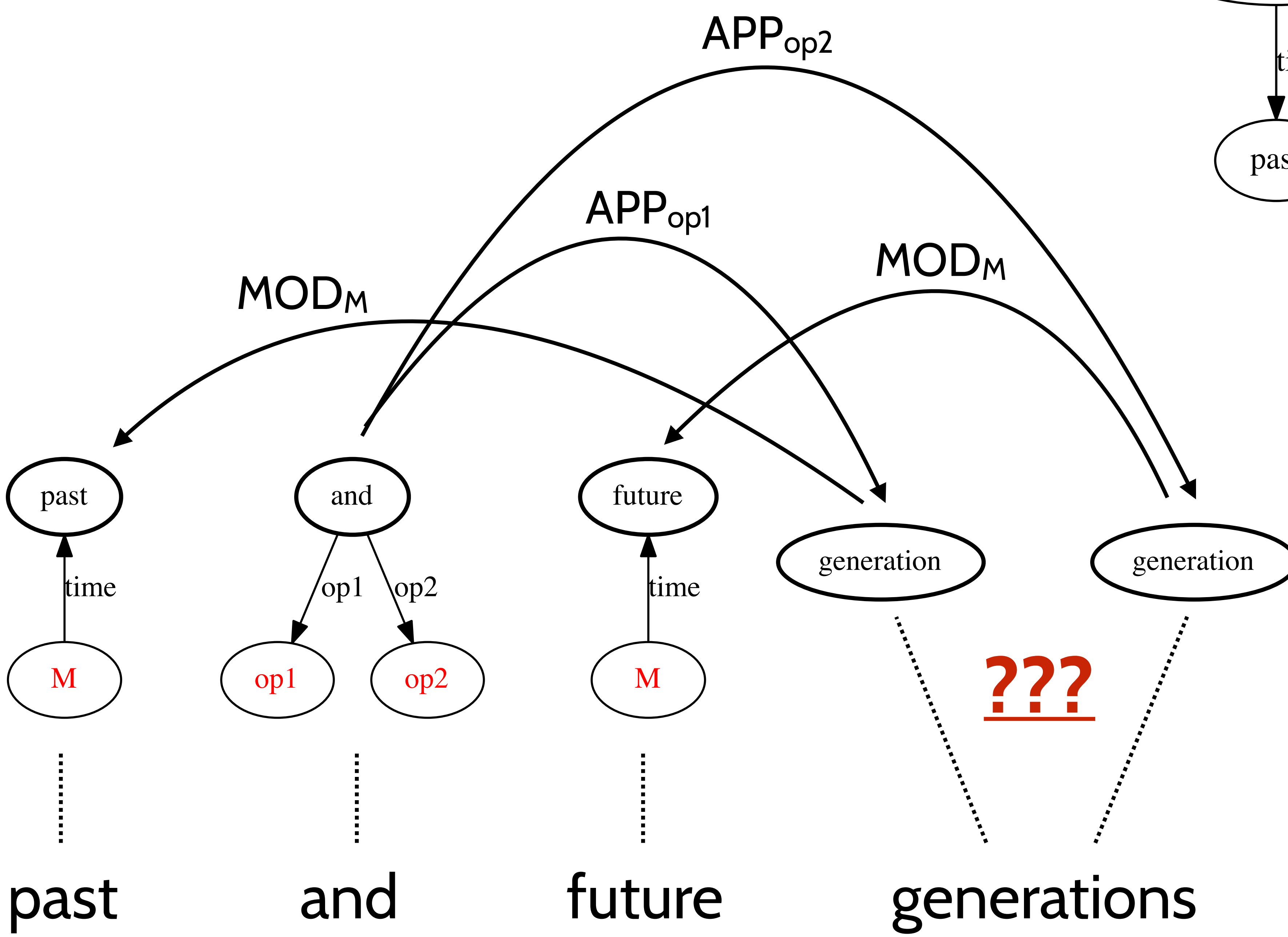
Problem 1: Coreference



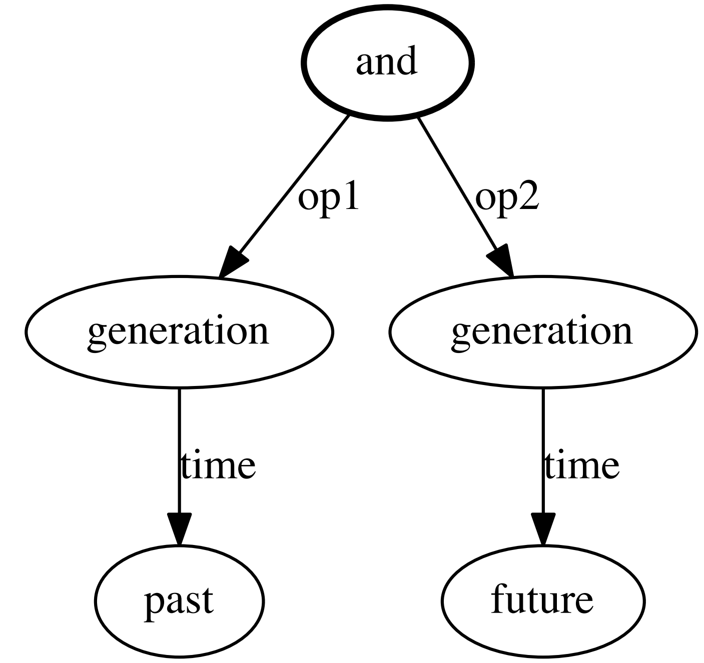
Unifying these is formally and practically challenging

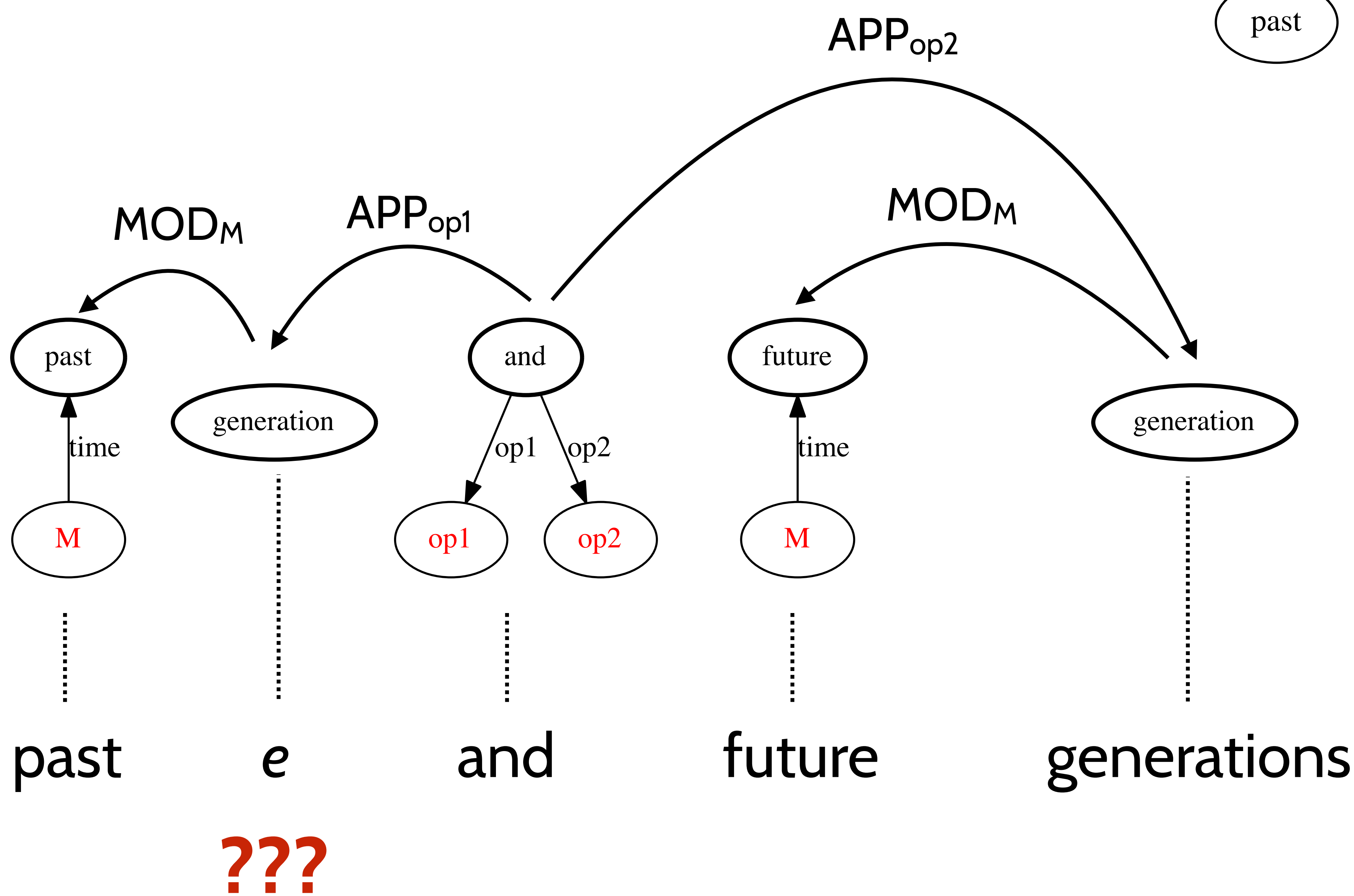
Problem 2: Multiplicity





Problem 2: Multiplicity





Conclusion / Future directions

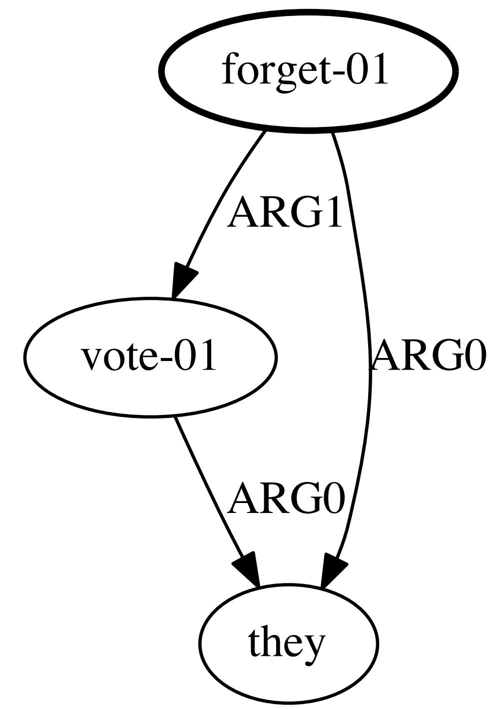
Conclusion / Future directions

- Semantic parsing with this method works very well in practice. Type information helps!
- Some open problems remain:
  - ellipsis
  - nested relative clauses have weird derivations
  - projectivity
  - AMR-specific issues such as coreference and unaligned nodes
- AMRs as a playground for semantic parsing

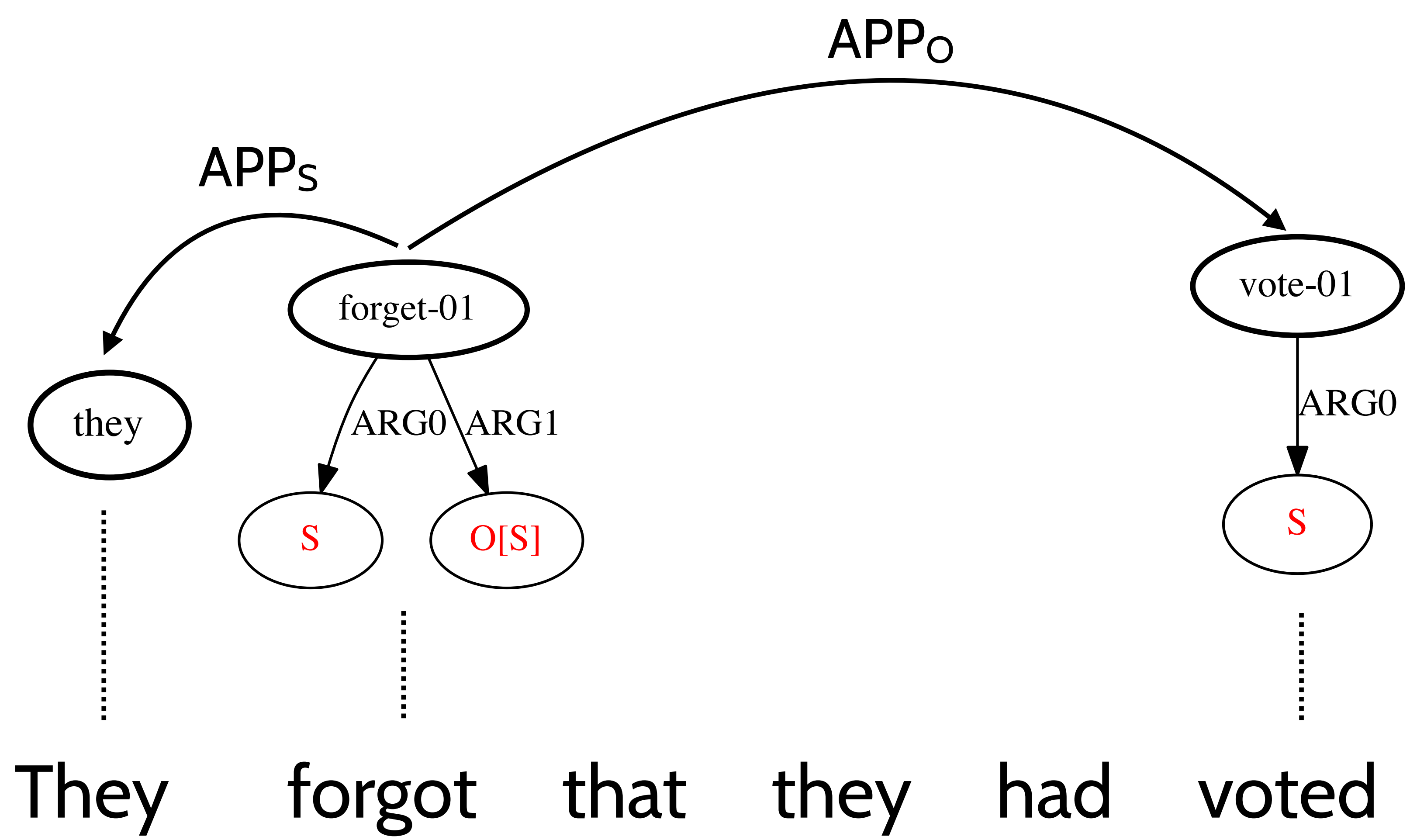
Conclusion / Future directions

- We approach dependency trees from the other side: AM dependency trees are *defined* to generate the semantics.
- Potential analogy: if AM dependency trees correspond to basic dependency trees, then AMRs correspond to enhanced dependency trees.

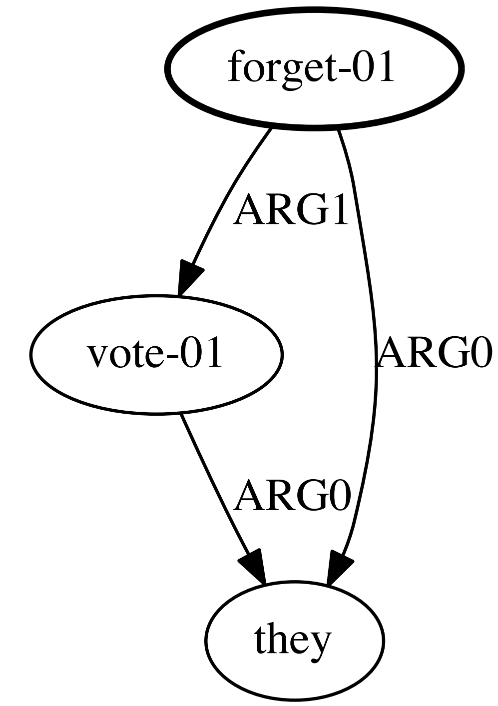
Problem 1: Coreference



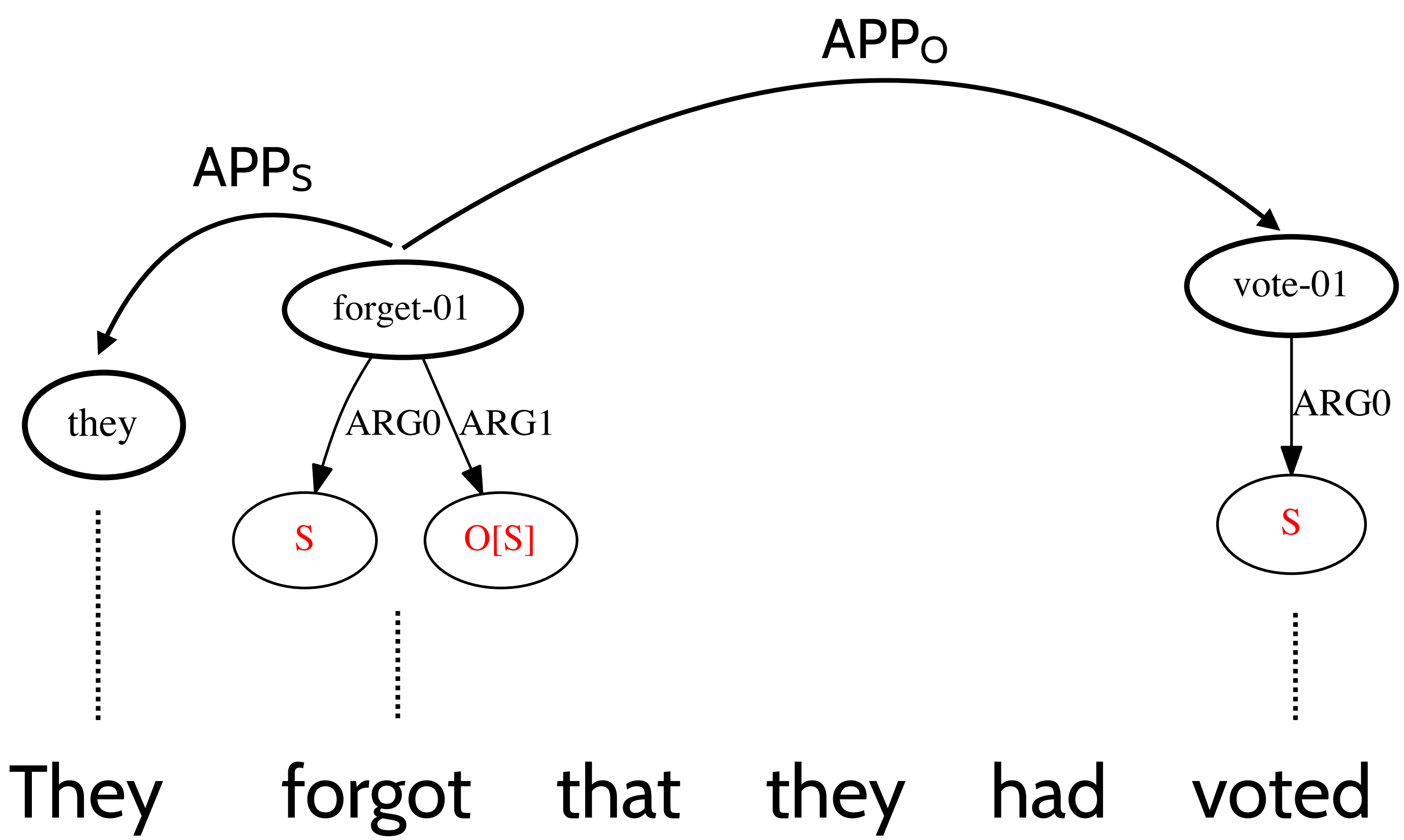
We can do this (and our system does):



Problem 1: Coreference



We can do this (and our system does):



But:

- Linguistically unsatisfying
- Does not work for longer range coreference