

Dependency Parsing and Feature-based Parsing

Ling 571 — Deep Processing Techniques for NLP

October 24, 2022

Shane Steinert-Threlkeld

Announcements

- HW3 reference code available
 - Sym-linked from hw4 directory (example_cky.py)
- HW4 notes on OOV: not necessary in base implementation; can be used as your improvement (for coverage)
- Mid-term feedback! <https://forms.gle/UGwT2ZFPbNNCvAis8>
 - Link also in Canvas announcement

Noun Phrase of the Week

A friend is apparently making "brown butter toffee pretzel chocolate chunk cookies", if anyone needs a delicious and chaotic noun phrase example

<https://twitter.com/EmmaSManning/status/1319750294666883075>

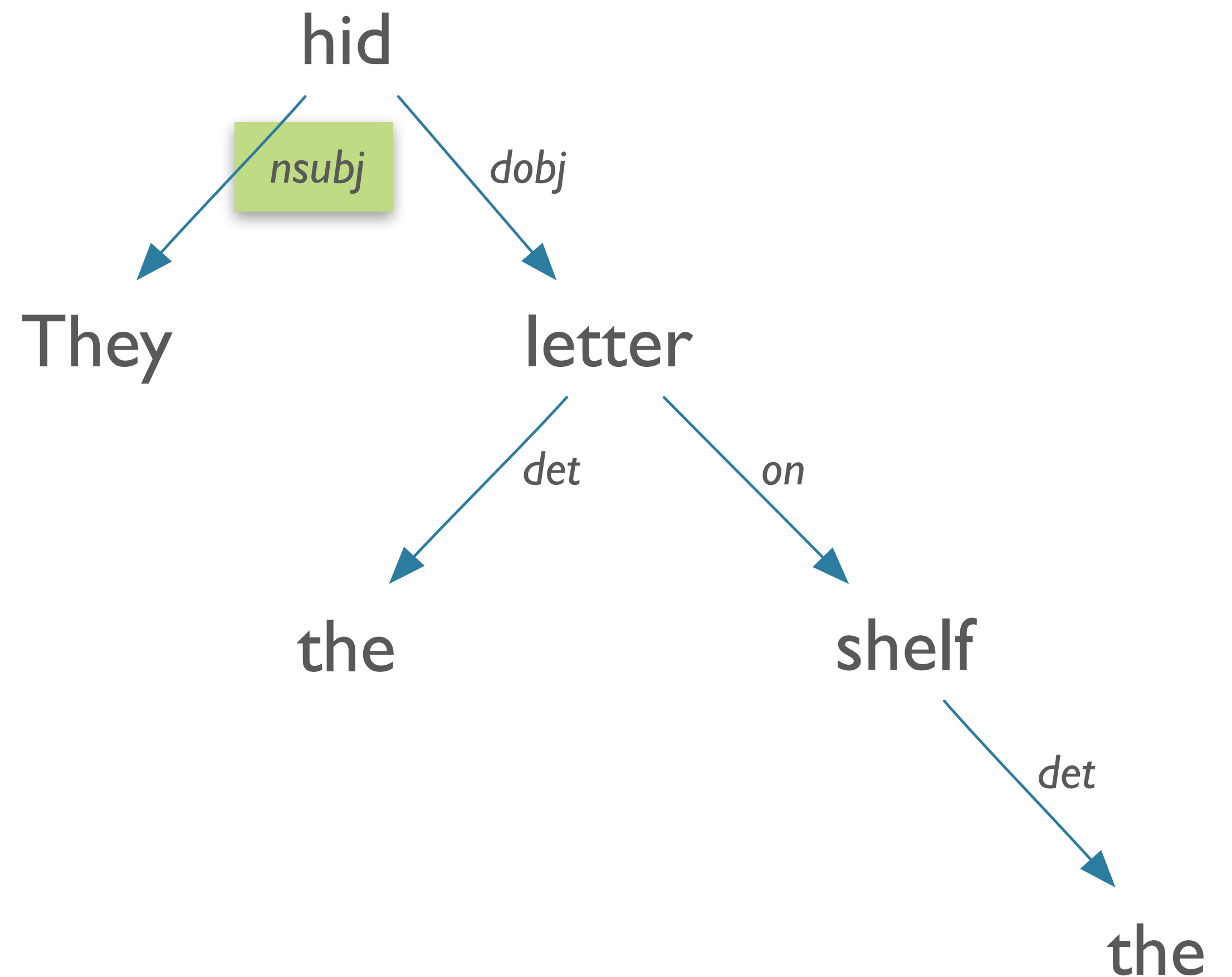
Today

- **Dependency Parsing**
 - Transition-based Parsing
- Feature-based Parsing
 - Motivation
 - Features
 - Unification

Dependency Parse Example:

They hid the letter on the shelf

| Argument Dependencies | |
|-----------------------|------------------------|
| Abbreviation | Description |
| nsubj | nominal subject |
| csbj | clausal subject |
| dobj | direct object |
| iobj | indirect object |
| pobj | object of preposition |
| Modifier Dependencies | |
| Abbreviation | Description |
| tmod | temporal modifier |
| appos | appositional modifier |
| det | determiner |
| prep | prepositional modifier |



Transition-Based Parsing

- Parsing defined in terms of sequence of transitions

Transition-Based Parsing

- Parsing defined in terms of sequence of transitions
- Alternative methods for learning/decoding
 - Most common model: Greedy classification-based approach
 - Very efficient: $O(n)$

Transition-Based Parsing

- Parsing defined in terms of sequence of transitions
- Alternative methods for learning/decoding
 - Most common model: Greedy classification-based approach
 - Very efficient: $O(n)$
- Best-known implementations:
 - Nivre's MALTParser
 - Nivre et al (2006); Nivre & Hall (2007)

Transition-Based Parsing

- A transition-based system for dependency parsing is:
 - A set of **configurations** C

Transition-Based Parsing

- A transition-based system for dependency parsing is:
 - A set of **configurations** C
 - A set of **transitions** between configurations

Transition-Based Parsing

- A transition-based system for dependency parsing is:
 - A set of **configurations** \mathcal{C}
 - A set of **transitions** between configurations
 - A transition function between configurations

Transition-Based Parsing

- A transition-based system for dependency parsing is:
 - A set of **configurations** \mathcal{C}
 - A set of **transitions** between configurations
 - A transition function between configurations
 - An initialization function (for \mathcal{C}_0)

Transition-Based Parsing

- A transition-based system for dependency parsing is:
 - A set of **configurations** \mathcal{C}
 - A set of **transitions** between configurations
 - A transition function between configurations
 - An initialization function (for \mathcal{C}_0)
 - A set of terminal configurations (“end states”)

Configurations

- A configuration for a sentence x is the triple (Σ, B, A) :
- Σ is a stack with elements corresponding to the nodes (words + ROOT) in x
- B (aka the buffer) is a list of nodes in x
- A is the set of dependency arcs in the analysis so far,
 - (w_i, L, w_j) , where w_x is a node in x and L is a dependency label

Transitions

- Transitions convert one configuration to another
 - $C_i = t(C_{i-1})$, where t is the transition
- Dependency graph for a sent:
 - The set of arcs resulting from a sequence of transitions
- The parse of the sentence is that resulting from the initial state through the sequence of transitions to a legal terminal state

Dependencies → Transitions

- To parse a sentence, we need the sequence of transitions that derives it

Dependencies → Transitions

- To parse a sentence, we need the sequence of transitions that derives it
- How can we determine sequence of transitions, given a parse?

Dependencies → Transitions

- To parse a sentence, we need the sequence of transitions that derives it
- How can we determine sequence of transitions, given a parse?
- This is defining our ***oracle*** function:
 - How to take a parse and translate it into a series of transitions

Dependencies → Transitions

- Many different oracles:
 - Nivre's arc-standard
 - Nivre's arc-eager
 - Non-projectivity with Attardi's
 - ...

Dependencies → Transitions

- Many different oracles:
 - Nivre's arc-standard
 - Nivre's arc-eager
 - Non-projectivity with Attardi's
 - ...
- Generally:
 - Use oracle to identify gold transitions
 - Train **classifier** to predict best transition in new config

Nivre's Arc-Standard Oracle

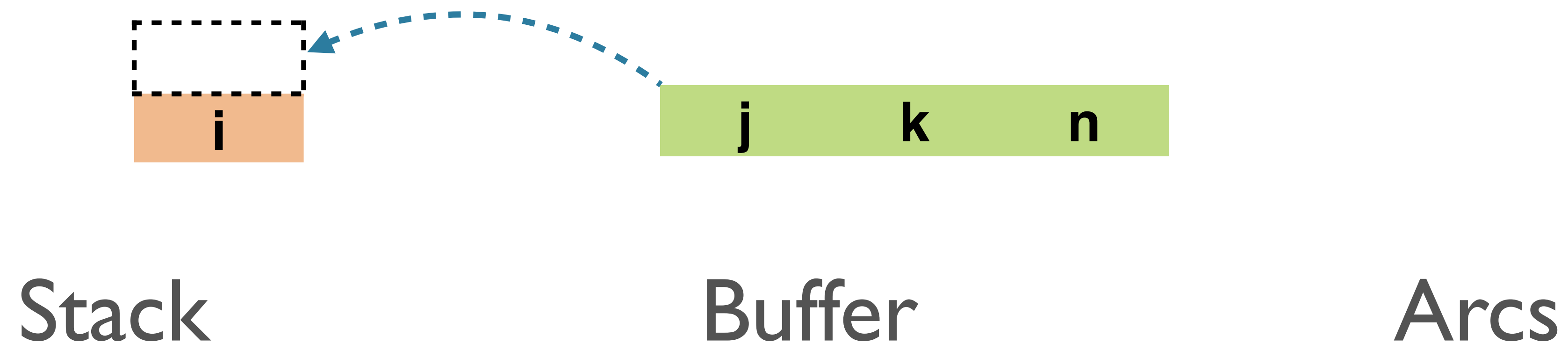
- Words: w_1, \dots, w_n
 - $w_0 = \text{ROOT}$
- Initialization:
 - $\text{Stack} = [w_0]; \text{Buffer} = [w_1, \dots, w_n]; \text{Arcs} = \emptyset$
- Termination:
 - $\text{Stack} = \sigma; \text{Buffer} = []; \text{Arcs} = A$
 - for any σ and A

Nivre's Arc-Standard Oracle

- Transitions are one of three:
 - Shift
 - Left-Arc
 - Right-Arc

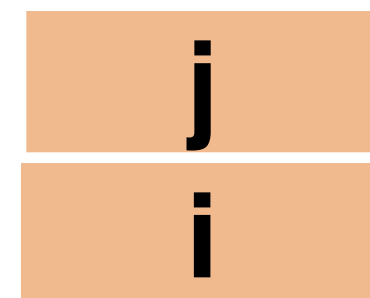
Transitions: Shift

- *Shift* first element of buffer to top of stack.
- $[i][j,k,n,\dots][\] \rightarrow [i,j][k,n,\dots][\]$



Transitions: Shift

- *Shift* first element of buffer to top of stack.
- $[i][j,k,n,\dots][\] \rightarrow [i,j][k,n,\dots][\]$



Stack

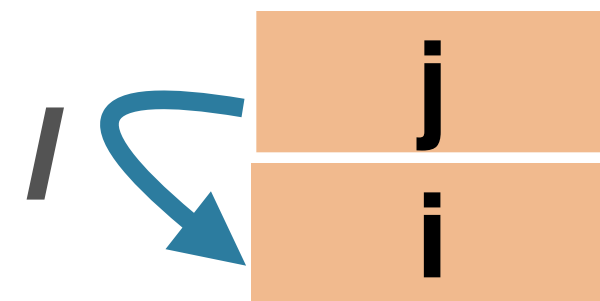


Buffer

Arcs

Transitions: Left-Arc

- Add arc from element at top of stack to **second element on stack** with dependency label **l**
- Pop **second element** from stack.
- $[i,j] [k,n,\dots] A \rightarrow [j] [k,n,\dots] A \cup [(j,l,i)]$



Stack



Buffer

Arcs

Transitions: Left-Arc

- Add arc from element at top of stack to **second element on stack** with dependency label **l**
- Pop **second element** from stack.
- $[i,j] [k,n,\dots] A \rightarrow [j] [k,n,\dots] A \cup [(j,l,i)]$

j

k

n

(j,l,i)

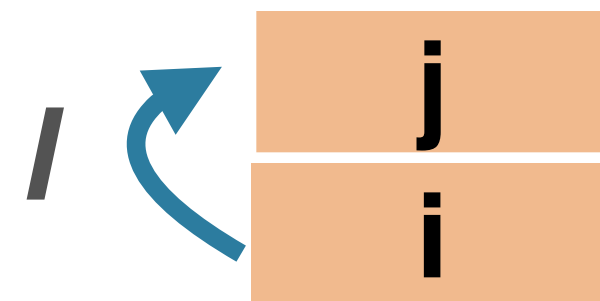
Stack

Buffer

Arcs

Transitions: Right-Arc

- Add arc from **second element on stack** to **top element on stack** with dependency label **l**
- Pop **top element** from stack.
- $[i,j] [k,n,\dots] A \rightarrow [j] [k,n,\dots] A \cup [(i,l,j)]$



Stack



Buffer

Arcs

Transitions: Right-Arc

- Add arc from **second element on stack** to **top element on stack** with dependency label **l**
- Pop **top element** from stack.
- $[i,j] [k,n,\dots] A \rightarrow [j] [k,n,\dots] A \cup [(i,l,j)]$

i

k

n

(i,l,j)

Stack

Buffer

Arcs

Training Process

- Each step of the algorithm is a decision point between the three states
- We want to train a model to decide between the three options at each step
 - (Reduce to a classification problem)
- We start with:
 - A treebank
 - An *oracle* process for guiding the transitions
 - A discriminative learner to relate the transition to features of the current configuration

Training Process, Formally:

(Σ, B, A)

- 1) $c \leftarrow c_0(S)$
- 2) **while** c is not terminal
- 3) $t \leftarrow o(c)$ # Choose the (o)ptimal transition for the config c
- 4) $c \leftarrow t(c)$ # Move to the next configuration
- 5) **return** G_c

Testing Process, Formally:

(Σ, B, A)

- 1) $c \leftarrow c_0(S)$
- 2) **while** c is not terminal
- 3) $t \leftarrow \lambda_c(c)$ # Choose the transition given model parameters at c
- 4) $c \leftarrow t(c)$ # Move to the next configuration
- 5) **return** G_c

Representing Configurations with Features

- **Address**

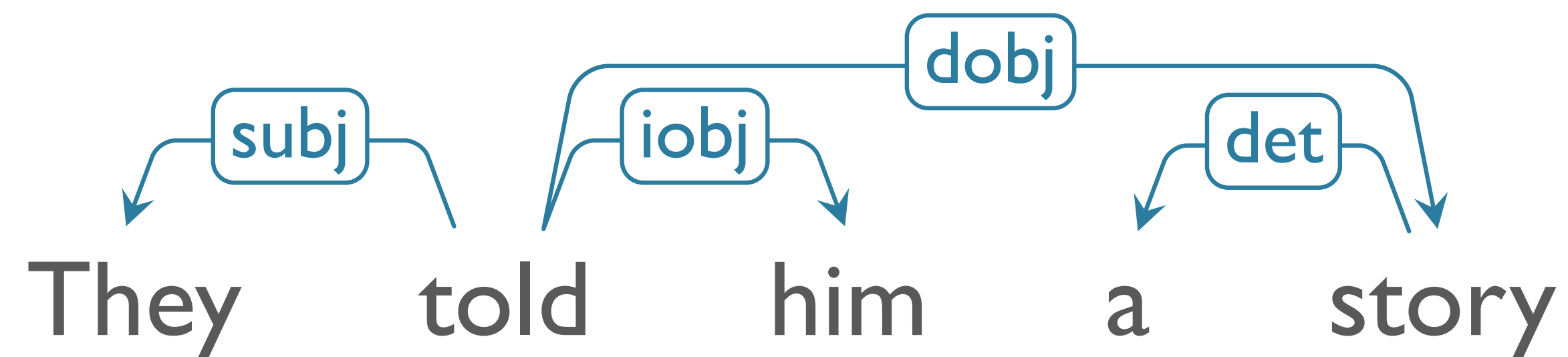
- Locate a given word:
 - By position in stack
 - By position in buffer
 - By attachment to a word in buffer

- **Attributes**

- Identity of word
- lemma for word
- POS tag of word
- Dependency label for word ← *conditioned on previous decisions!*

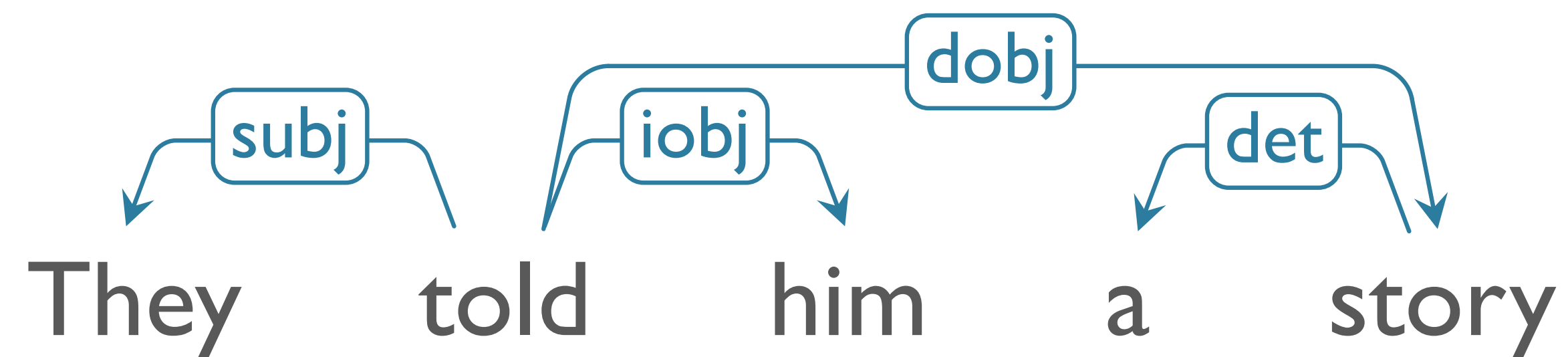
Example:

| Action | Stack | Buffer |
|--------|--------|-------------------------|
| | [ROOT] | [They told him a story] |



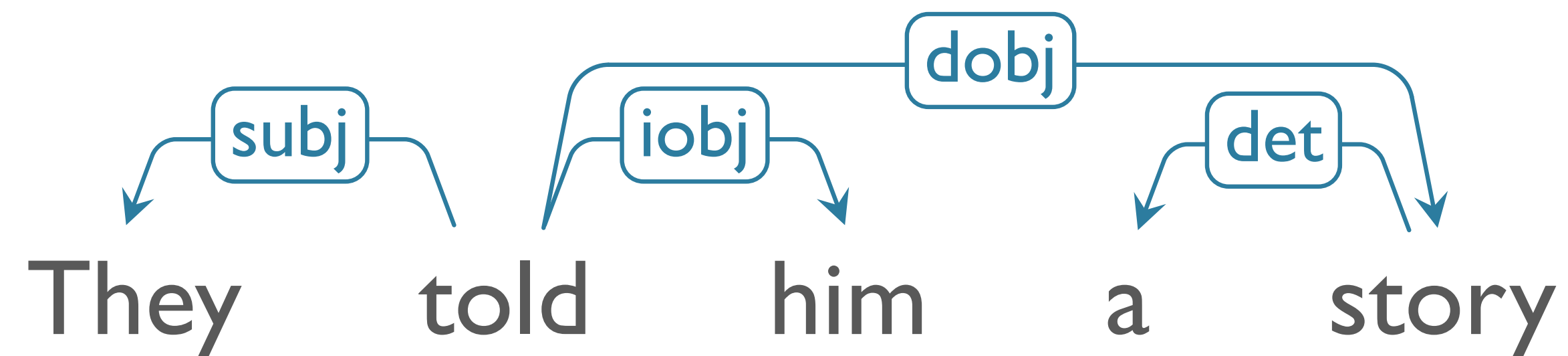
Example:

| Action | Stack | Buffer |
|--------|--------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |



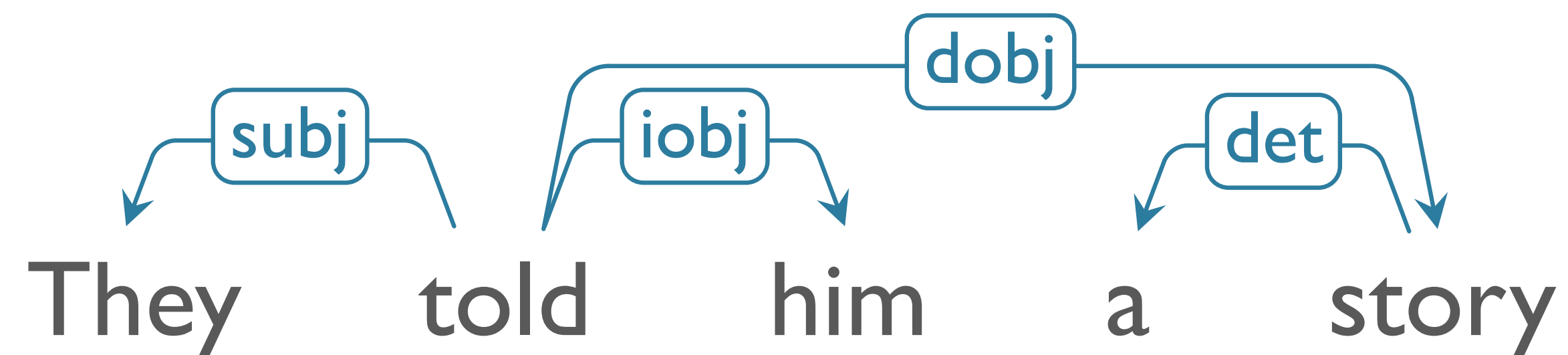
Example:

| Action | Stack | Buffer |
|--------|--------------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |
| Shift | [ROOT, They, told] | [him a story] |



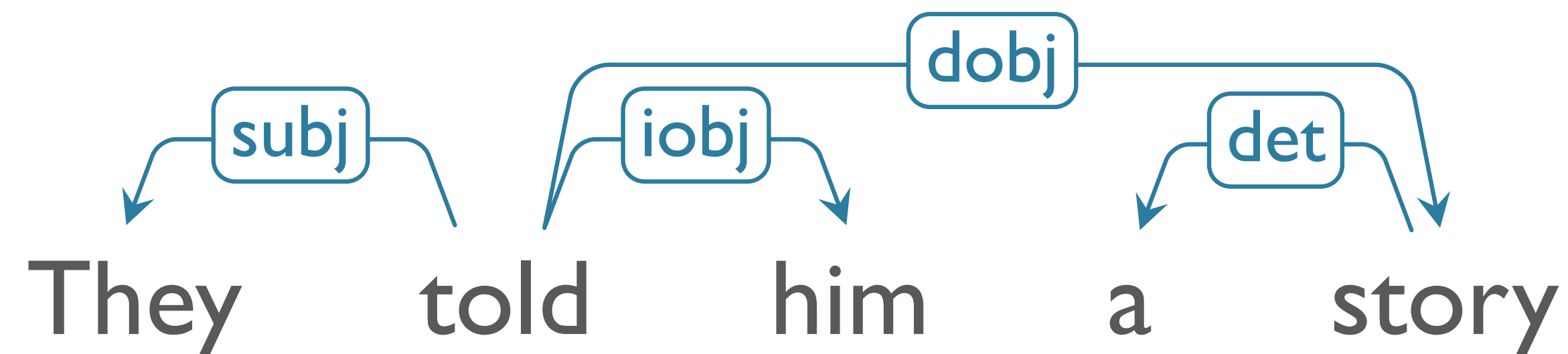
Example:

| Action | Stack | Buffer |
|-----------------|--------------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |
| Shift | [ROOT, They, told] | [him a story] |
| Left-Arc (subj) | [ROOT, told] | [him a story] |



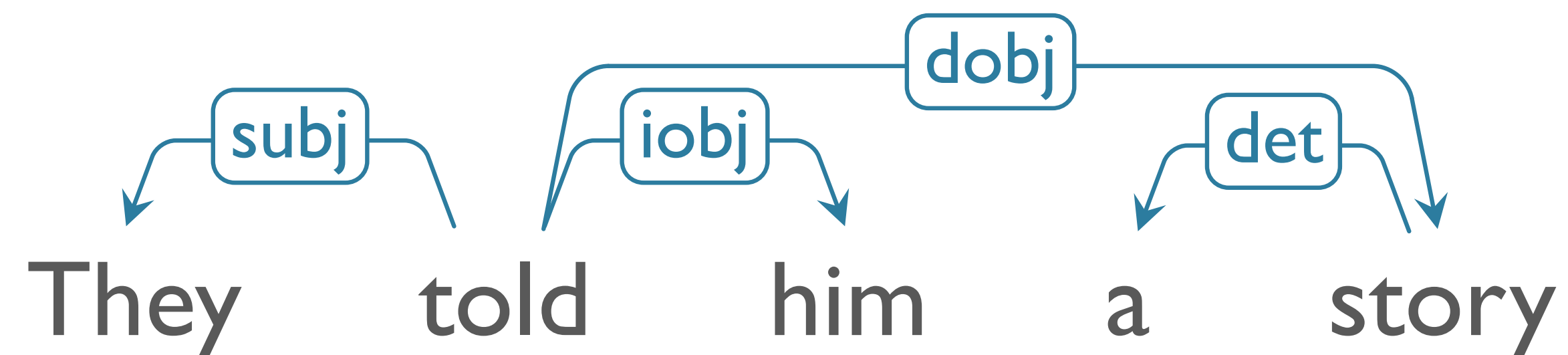
Example:

| Action | Stack | Buffer |
|-----------------|--------------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |
| Shift | [ROOT, They, told] | [him a story] |
| Left-Arc (subj) | [ROOT, told] | [him a story] |
| Shift | [ROOT, told, him] | [a story] |



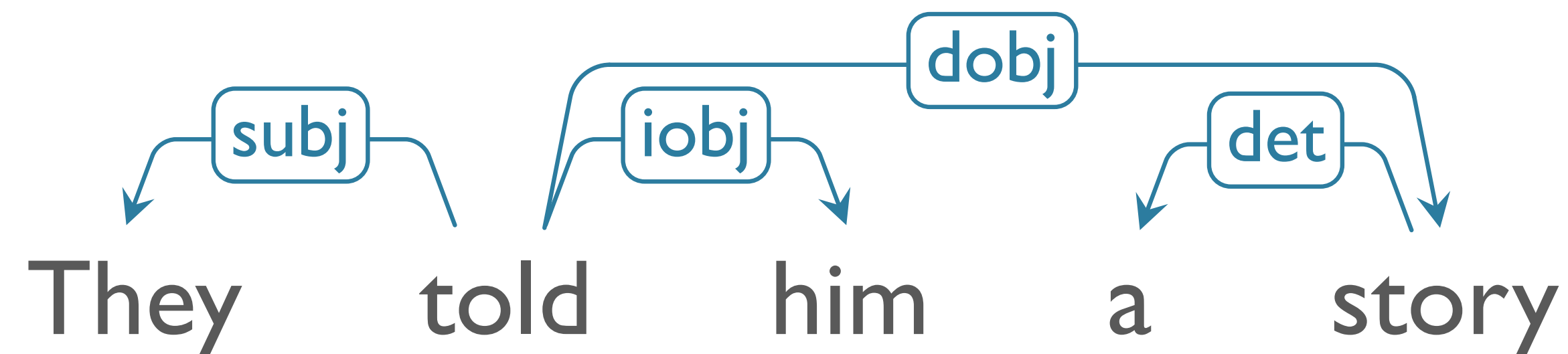
Example:

| Action | Stack | Buffer |
|------------------|--------------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |
| Shift | [ROOT, They, told] | [him a story] |
| Left-Arc (subj) | [ROOT, told] | [him a story] |
| Shift | [ROOT, told, him] | [a story] |
| Right-Arc (iobj) | [ROOT, told] | [a story] |



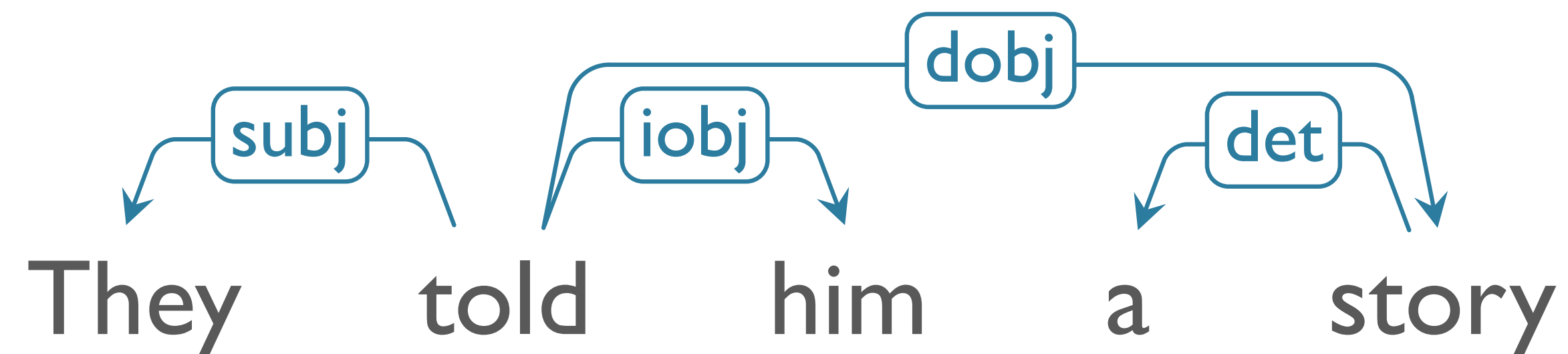
Example:

| Action | Stack | Buffer |
|------------------|--------------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |
| Shift | [ROOT, They, told] | [him a story] |
| Left-Arc (subj) | [ROOT, told] | [him a story] |
| Shift | [ROOT, told, him] | [a story] |
| Right-Arc (iobj) | [ROOT, told] | [a story] |
| Shift | [ROOT, told, a] | [story] |



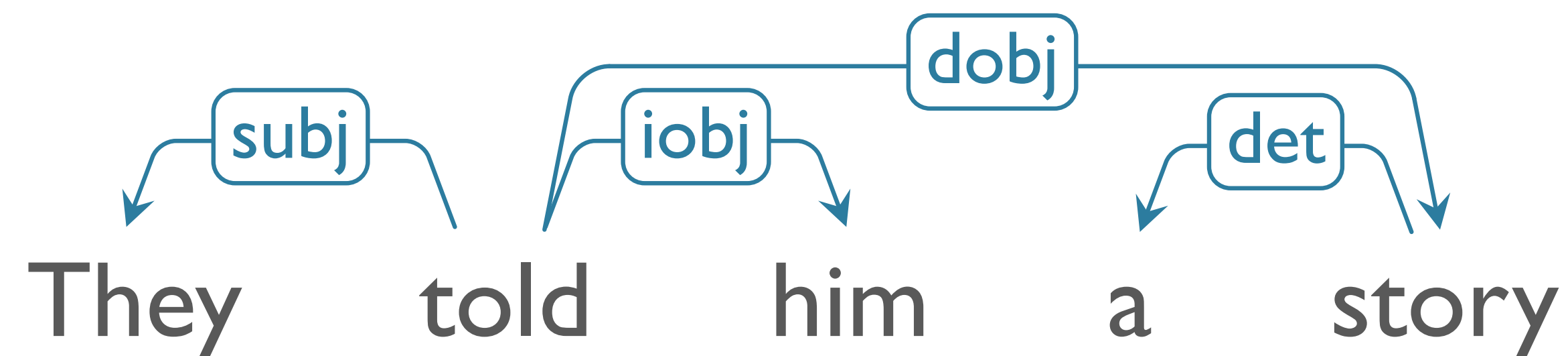
Example:

| Action | Stack | Buffer |
|------------------|-----------------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |
| Shift | [ROOT, They, told] | [him a story] |
| Left-Arc (subj) | [ROOT, told] | [him a story] |
| Shift | [ROOT, told, him] | [a story] |
| Right-Arc (iobj) | [ROOT, told] | [a story] |
| Shift | [ROOT, told, a] | [story] |
| Shift | [ROOT,told, a, story] | [] |



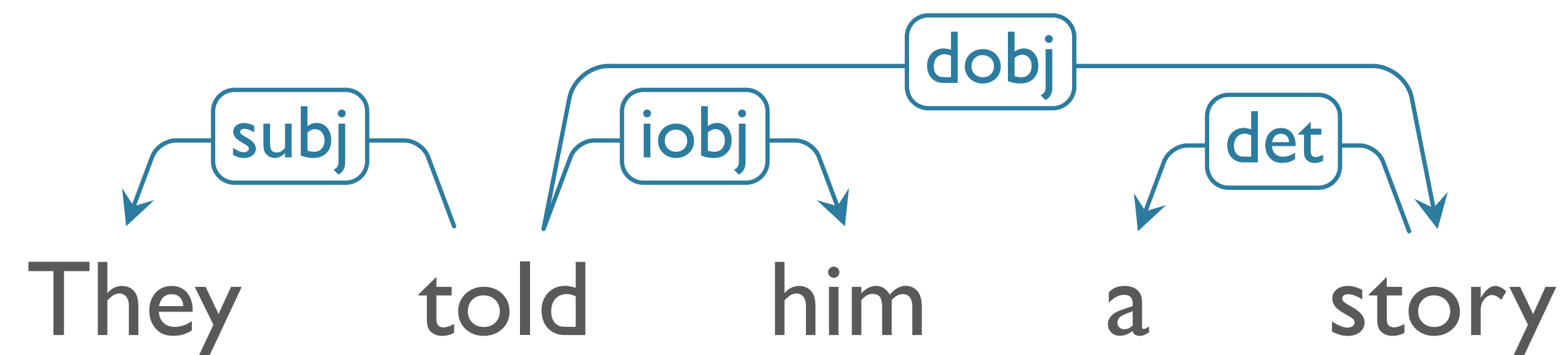
Example:

| Action | Stack | Buffer |
|------------------|-----------------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |
| Shift | [ROOT, They, told] | [him a story] |
| Left-Arc (subj) | [ROOT, told] | [him a story] |
| Shift | [ROOT, told, him] | [a story] |
| Right-Arc (iobj) | [ROOT, told] | [a story] |
| Shift | [ROOT, told, a] | [story] |
| Shift | [ROOT,told, a, story] | [] |
| Left-Arc (Det) | [ROOT, told, story] | [] |



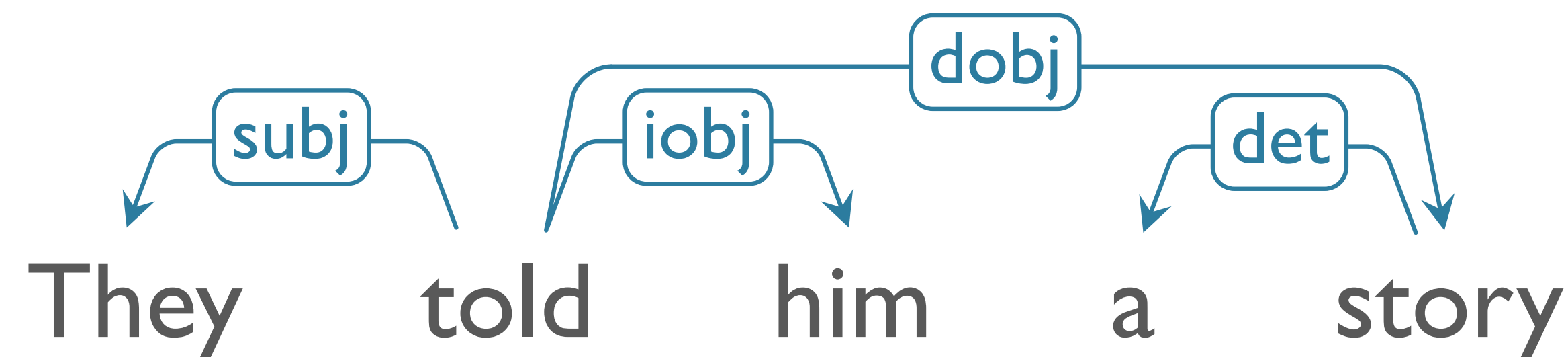
Example:

| Action | Stack | Buffer |
|------------------|-----------------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |
| Shift | [ROOT, They, told] | [him a story] |
| Left-Arc (subj) | [ROOT, told] | [him a story] |
| Shift | [ROOT, told, him] | [a story] |
| Right-Arc (iobj) | [ROOT, told] | [a story] |
| Shift | [ROOT, told, a] | [story] |
| Shift | [ROOT,told, a, story] | [] |
| Left-Arc (Det) | [ROOT, told, story] | [] |
| Right-Arc (dobj) | [ROOT, told] | [] |



Example:

| Action | Stack | Buffer |
|------------------|-----------------------|-------------------------|
| | [ROOT] | [They told him a story] |
| Shift | [ROOT, They] | [told him a story] |
| Shift | [ROOT, They, told] | [him a story] |
| Left-Arc (subj) | [ROOT, told] | [him a story] |
| Shift | [ROOT, told, him] | [a story] |
| Right-Arc (iobj) | [ROOT, told] | [a story] |
| Shift | [ROOT, told, a] | [story] |
| Shift | [ROOT,told, a, story] | [] |
| Left-Arc (Det) | [ROOT, told, story] | [] |
| Right-Arc (dobj) | [ROOT, told] | [] |
| Right-Arc (root) | [ROOT] | [] |



Transition-Based Parsing Summary

- *Shift-Reduce* [reduce = pop] paradigm, bottom-up approach
- **Pros:**
 - Single pass, $O(n)$ complexity
 - Reduce parsing to classification problem; easy to introduce new features
- **Cons:**
 - Only makes local decisions, may not find global optimum
 - Does not handle non-projective trees without hacks
 - e.g. transforming nonprojective trees to projective in training data; reconverting after

Other Notes

- ...is this a parser?
 - No, not really!
 - Transforms problem into sequence labeling task, of a sort.
 - e.g. (SH, LA, SH, RA, SH, SH, LA, RA)
 - Sequence score is sum of transition scores

Other Notes

- Classifier: Any
 - Originally, SVMs
 - Currently: NNs (LSTMs, pre-trained Transformer-based)
- State-of-the-art: UAS: 97.2%; LAS: 95.7%
- http://nlpprogress.com/english/dependency_parsing.html

Dependency parsing

Dependency parsing is the task of extracting a dependency parse of a sentence that represents its grammatical structure and defines the relationships between “head” words and words, which modify those heads.

Example:

```
      root
      |
      | +-----dobj-----+
      | |                   |
nsubj | | +-----det-----+ | +-----nmod-----+
+---+ | | | +-----nmod--+ | | +-----case--+ |
| | | | | +-----+ | | | +-----+ |
+ | + | + + + | + | + + | + | + |
I prefer the morning flight through Denver
```

Relations among the words are illustrated above the sentence with directed, labeled arcs from heads to dependents (+ indicates the dependent).

Penn Treebank

Models are evaluated on the [Stanford Dependency](#) conversion (v3.3.0) of the Penn Treebank with **predicted** POS-tags. Punctuation symbols are excluded from the evaluation. Evaluation metrics are unlabeled attachment score (UAS) and labeled attachment score (LAS). UAS does not consider the semantic relation (e.g. Subj) used to label the attachment between the head and the child, while LAS requires a semantic correct label for each attachment. Here, we also mention the predicted POS tagging accuracy.

| Model | POS | UAS | LAS | Paper / Source | Code |
|---|-------|-------|-------|---|--------------------------|
| HPSG Parser (Joint) + XLNet (Zhou and Zhao, 2019) | 97.3 | 97.20 | 95.72 | Head-Driven Phrase Structure Grammar Parsing on Penn Treebank | Official |
| HPSG Parser (Joint) + BERT (Zhou and Zhao, 2019) | 97.3 | 97.00 | 95.43 | Head-Driven Phrase Structure Grammar Parsing on Penn Treebank | Official |
| CVT + Multi-Task (Clark et al., 2018) | 97.74 | 96.61 | 95.02 | Semi-Supervised Sequence Modeling with Cross-View Training | Official |
| Graph-based parser with GNNs (Ji et al., 2019) | 97.3 | 95.97 | 94.31 | Graph-based Dependency Parsing with Graph Neural Networks | |
| Deep Biaffine (Dozat and Manning, 2017) | 97.3 | 95.74 | 94.08 | Deep Biaffine Attention for Neural Dependency Parsing | Official |
| jPTDP (Nguyen and Verspoor, 2018) | 97.97 | 94.51 | 92.87 | An improved neural network model for joint POS tagging and dependency parsing | Official |
| Andor et al. (2016) | 97.44 | 94.61 | 92.79 | Globally Normalized Transition-Based Neural Networks | |

Other Notes

- Classifier: Any
 - Originally, SVMs
 - Currently: NNs (LSTMs, pre-trained Transformer-based)
- State-of-the-art: UAS: 97.2%; LAS: 95.7%
- http://nlpprogress.com/english/dependency_parsing.html

Story time! 

Dependency parsing

Dependency parsing is the task of extracting a dependency parse of a sentence that represents its grammatical structure and defines the relationships between “head” words and words, which modify those heads.

Example:

```
graph TD
    root -->|+-----dobj-----+| flight
    root -->|nsubj| I
    root -->|+-----det-----+| the
    root -->|+-----nmod-----+| through
    root -->|+-----case-----+| Denver
    root -->|+-----mod-----+| morning
    root -->|+-----adv-----+| prefer
    root -->|+-----adv-----+| flight
```

Relations among the words are illustrated above the sentence with directed, labeled arcs from heads to dependents (+ indicates the dependent).

Penn Treebank

Models are evaluated on the [Stanford Dependency](#) conversion (v3.3.0) of the Penn Treebank with **predicted** POS-tags. Punctuation symbols are excluded from the evaluation. Evaluation metrics are unlabeled attachment score (UAS) and labeled attachment score (LAS). UAS does not consider the semantic relation (e.g. Subj) used to label the attachment between the head and the child, while LAS requires a semantic correct label for each attachment. Here, we also mention the predicted POS tagging accuracy.

| Model | POS | UAS | LAS | Paper / Source | Code |
|---|-------|-------|-------|---|--------------------------|
| HPSG Parser (Joint) + XLNet (Zhou and Zhao, 2019) | 97.3 | 97.20 | 95.72 | Head-Driven Phrase Structure Grammar Parsing on Penn Treebank | Official |
| HPSG Parser (Joint) + BERT (Zhou and Zhao, 2019) | 97.3 | 97.00 | 95.43 | Head-Driven Phrase Structure Grammar Parsing on Penn Treebank | Official |
| CVT + Multi-Task (Clark et al., 2018) | 97.74 | 96.61 | 95.02 | Semi-Supervised Sequence Modeling with Cross-View Training | Official |
| Graph-based parser with GNNs (Ji et al., 2019) | 97.3 | 95.97 | 94.31 | Graph-based Dependency Parsing with Graph Neural Networks | |
| Deep Biaffine (Dozat and Manning, 2017) | 97.3 | 95.74 | 94.08 | Deep Biaffine Attention for Neural Dependency Parsing | Official |
| jPTDP (Nguyen and Verspoor, 2018) | 97.97 | 94.51 | 92.87 | An improved neural network model for joint POS tagging and dependency parsing | Official |
| Andor et al. (2016) | 97.44 | 94.61 | 92.79 | Globally Normalized Transition-Based Neural Networks | |

Parsey McParseface

Parsey McParseface



The latest news from Google AI

Announcing SyntaxNet: The World's Most Accurate Parser Goes Open Source

Thursday, May 12, 2016

Posted by Slav Petrov, Senior Staff Research Scientist

At Google, we spend a lot of time thinking about how [computer systems](#) can [read](#) and [understand human language](#) in order [to process it](#) in [intelligent ways](#). Today, we are excited to share the fruits of our research with the broader community by releasing [SyntaxNet](#), an open-source neural network framework implemented in [TensorFlow](#) that provides a foundation for [Natural Language Understanding](#) (NLU) systems. Our release includes all the code needed to train new SyntaxNet models on your own data, as well as *Parsey McParseface*, an English parser that we have trained for you and that you can use to analyze English text.

Parsey McParseface is built on powerful machine learning algorithms that learn to analyze the linguistic structure of language, and that can explain the functional role of each word in a given sentence. Because Parsey McParseface is the [most accurate such model in the world](#), we hope that it will be useful to developers and researchers interested in automatic extraction of information, translation, and other core applications of NLU.

<https://ai.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html>

Parsey McParseface



The latest news from Google AI

Announcing SyntaxNet: The World's Most Accurate Parser Goes Open Source

Thursday, May 12, 2016

Posted by Slav Petrov, Senior Staff Research Scientist

At Google, we spend a lot of time thinking about how [computer systems](#) can [read](#) and [understand human language](#) in order [to process it](#) in [intelligent ways](#). Today, we are excited to share the fruits of our research with the broader community by releasing [SyntaxNet](#), an open-source neural network framework implemented in [TensorFlow](#) that provides a foundation for [Natural Language Understanding](#) (NLU) systems. Our release includes all the code needed to train new SyntaxNet models on your own data, as well as [Parsey McParseface](#), an English parser that we have trained for you and that you can use to analyze English text.

Parsey McParseface is built on powerful machine learning algorithms that learn to analyze the linguistic structure of language, and that can explain the functional role of each word in a given sentence. Because Parsey McParseface is the [most accurate such model in the world](#), we hope that it will be useful to developers and researchers interested in automatic extraction of information, translation, and other core applications of NLU.

<https://ai.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html>

Parsey McParseface



SCI-TECH

Don't laugh: Google's Parsey McParseface is a serious IQ boost for computers



Google is giving away the tool it uses to understand language, Parsey McParseface

Okay, Google. Okay. We get it.

By Dieter Bohn | @backlon | May 12, 2016, 3:00pm EDT



Share

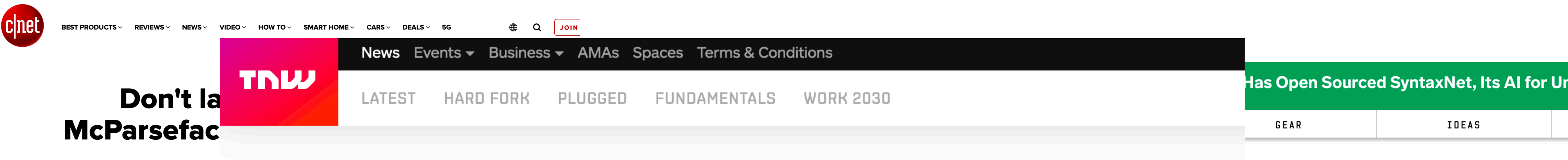
f SHARE

tweet TWEET

CADE METZ BUSINESS 05.12.16 03:00 PM

Google Has Open Sourced SyntaxNet, Its AI for Understanding Language

Parsey McParseface



SS 05.12.16 03:00 PM

THE VERGE

GOOGLE TECH

Google is giving us a way to understand

Okay, Google. Okay. We go

By Dieter Bohn | @backlon | May 12, 2016, 3:00pm EDT

Google just open sourced something called 'Parsey McParseface,' and it could change AI forever



by NATE SWANNER — May 12, 2016 in DESIGN & DEV

Has Open Sourced SyntaxNet, Its AI for Understanding Language

Parsey McParseface

Globally Normalized Transition-Based Neural Networks

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn,
Alessandro Presta, Kuzman Ganchev, Slav Petrov and Michael Collins*

Google Inc
New York, NY

{andor,chrisalberti,djweiss,severyn,apresta,kuzman,slav,mjcollins}@google.com

Abstract

We introduce a globally normalized transition-based neural network model that achieves state-of-the-art part-of-speech tagging, dependency parsing and sentence compression results. Our model is a simple feed-forward neural network that operates on a task-specific transition system, yet achieves comparable or better accuracies than recurrent models. We discuss the importance of global as opposed to local normalization: a key insight is that the label bias problem implies that globally normalized models can be strictly more expressive than locally normalized models.

Chen and Manning (2014). We do not use any recurrence, but perform beam search for maintaining multiple hypotheses and introduce global normalization with a conditional random field (CRF) objective (Bottou et al., 1997; Le Cun et al., 1998; Lafferty et al., 2001; Collobert et al., 2011) to overcome the label bias problem that locally normalized models suffer from. Since we use beam inference, we approximate the partition function by summing over the elements in the beam, and use early updates (Collins and Roark, 2004; Zhou et al., 2015). We compute gradients based on this approximate global normalization and perform full backpropagation training of all neural network parameters based on the CRF loss.

In Section 3 we revisit the label bias problem and the implication that globally normalized mod-

Parsey McParseface

Globally Normalized Transition-Based Neural Networks

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn,
Alessandro Presta, Kuzman Ganchev, Slav Petrov and Michael Collins*

Google Inc
New York, NY

{andor,chrisalberti,djweiss,severyn,apresta,kuzman,slav,mjcollins}@google.com

Abstract

We introduce a globally normalized transition-based neural network model that achieves state-of-the-art part-of-speech tagging, dependency parsing and sentence compression results. Our model is a simple feed-forward neural network that operates on a task-specific transition system, yet achieves comparable or better accuracies than recurrent models. We discuss the importance of global as opposed to local normalization: a key insight is that the label bias problem implies that globally normalized models can be strictly more expressive than locally normalized models.

Chen and Manning (2014). We do not use any recurrence, but perform beam search for maintaining multiple hypotheses and introduce global normalization with a conditional random field (CRF) objective (Bottou et al., 1997; Le Cun et al., 1998; Lafferty et al., 2001; Collobert et al., 2011) to overcome the label bias problem that locally normalized models suffer from. Since we use beam inference, we approximate the partition function by summing over the elements in the beam, and use early updates (Collins and Roark, 2004; Zhou et al., 2015). We compute gradients based on this approximate global normalization and perform full backpropagation training of all neural network parameters based on the CRF loss.

In Section 3 we revisit the label bias problem and the implication that globally normalized mod-

Parsey McParseface

Globally Normalized Transition-Based Neural Networks

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn,
Alessandro Presta, Kuzman Ganchev, Slav Petrov and Michael Collins*

Google Inc
New York, NY

{andor,chrisalberti,djweiss,severyn,apresta,kuzman,slav,mjcollins}@google.com

Abstract

We introduce a globally normalized transition-based neural network model that achieves state-of-the-art part-of-speech tagging, dependency parsing and sentence compression results. Our model is a simple feed-forward neural network that operates on a task-specific transition system, yet achieves comparable or better accuracies than recurrent models. We discuss the importance of global as opposed to local normalization: a key insight is that the label bias problem implies that globally normalized models can be strictly more expressive than locally normalized models.

Chen and Manning (2014). We do not use any recurrence, but perform beam search for maintaining multiple hypotheses and introduce global normalization with a conditional random field (CRF) objective (Bottou et al., 1997; Le Cun et al., 1998; Lafferty et al., 2001; Collobert et al., 2011) to overcome the label bias problem that locally normalized models suffer from. Since we use beam inference, we approximate the partition function by summing over the elements in the beam, and use early updates (Collins and Roark, 2004; Zhou et al., 2015). We compute gradients based on this approximate global normalization and perform full backpropagation training of all neural network parameters based on the CRF loss.

In Section 3 we revisit the label bias problem and the implication that globally normalized mod-

Great paper

Many methodological lessons on how to improve transition-based dependency parsing

BUT: don't believe (or at least beware) the hype!

Dependency Parsing: Summary

- Dependency Grammars:
 - Compactly represent pred–arg structure
 - Lexicalized, localized
 - Natural handling of flexible word order

Dependency Parsing: Summary

- Dependency Grammars:
 - Compactly represent pred–arg structure
 - Lexicalized, localized
 - Natural handling of flexible word order
- Dependency parsing:
 - Conversion to phrase structure trees
 - Graph-based parsing (MST), efficient non-proj $O(n^2)$
 - Transition-based parser
 - MALTparser: very efficient $O(n)$
 - Optimizes local decisions based on many rich features

Roadmap

- Dependency Parsing
 - Transition-based Parsing
- **Feature-based Parsing**
 - Motivation
 - Features
 - Unification

Feature-Based Parsing

Constraints & Compactness

- $S \rightarrow NP VP$
 - *They run.*
 - *He runs.*

Constraints & Compactness

- $S \rightarrow NP VP$
 - *They run.*
 - *He runs.*
- **But...**
 - **They runs*
 - **He run*
 - **He disappeared the flight*
- Violate agreement (number/person),
subcategorization -> over-generation

Enforcing Constraints with CFG Rules

- Agreement
 - $S \rightarrow NP_{sg+3p} VP_{sg+3p}$
 - $S \rightarrow NP_{pl+3p} VP_{pl+3p}$

Enforcing Constraints with CFG Rules

- Agreement
 - $S \rightarrow NP_{sg+3p} VP_{sg+3p}$
 - $S \rightarrow NP_{pl+3p} VP_{pl+3p}$
- Subcategorization:
 - $VP \rightarrow V_{transitive} NP$
 - $VP \rightarrow V_{intransitive}$
 - $VP \rightarrow V_{ditransitive} NP NP$
- Explosive, and loses key generalizations

Feature Grammars

- Need compact, general constraint

Feature Grammars

- Need compact, general constraint
- $S \rightarrow NP VP$ [iff NP and VP agree]

Feature Grammars

- Need compact, general constraint
- $S \rightarrow NP VP$ [iff NP and VP agree]
- How can we describe agreement & subcategory?
 - Decompose into elementary features that must be consistent
 - e.g. Agreement on number, person, gender, etc

Feature Grammars

- Need compact, general constraint
- $S \rightarrow NP VP$ [iff NP and VP agree]
- How can we describe agreement & subcategory?
 - Decompose into elementary features that must be consistent
 - e.g. Agreement on number, person, gender, etc
- Augment CF rules with feature constraints
 - Develop mechanism to enforce consistency
 - Elegant, compact, rich representation

Feature Representations

- Fundamentally **Attribute-Value** pairs
 - Values may be symbols or feature structures
 - Feature path: list of features in structure to value
 - “Reentrant feature structure” — sharing a structure
- Represented as
 - Attribute-Value Matrix (AVM)
 - Directed Acyclic Graph (DAG)

Attribute-Value Matrices (AVMs)

$$\begin{bmatrix} \text{ATTRIBUTE}_1 & \text{value}_1 \\ \text{ATTRIBUTE}_2 & \text{value}_2 \\ \vdots & \\ \text{ATTRIBUTE}_n & \text{value}_n \end{bmatrix}$$

AVM Examples

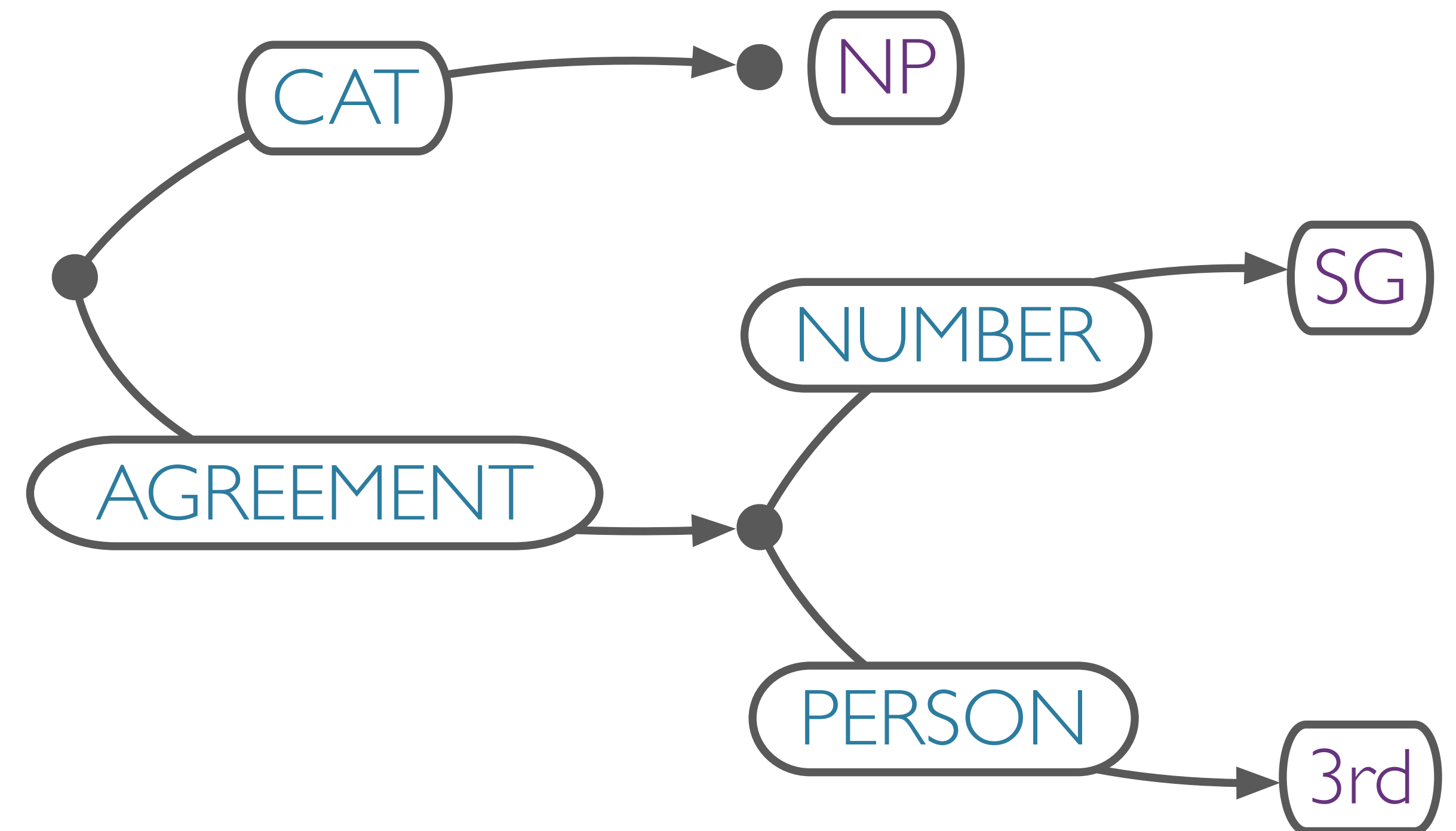
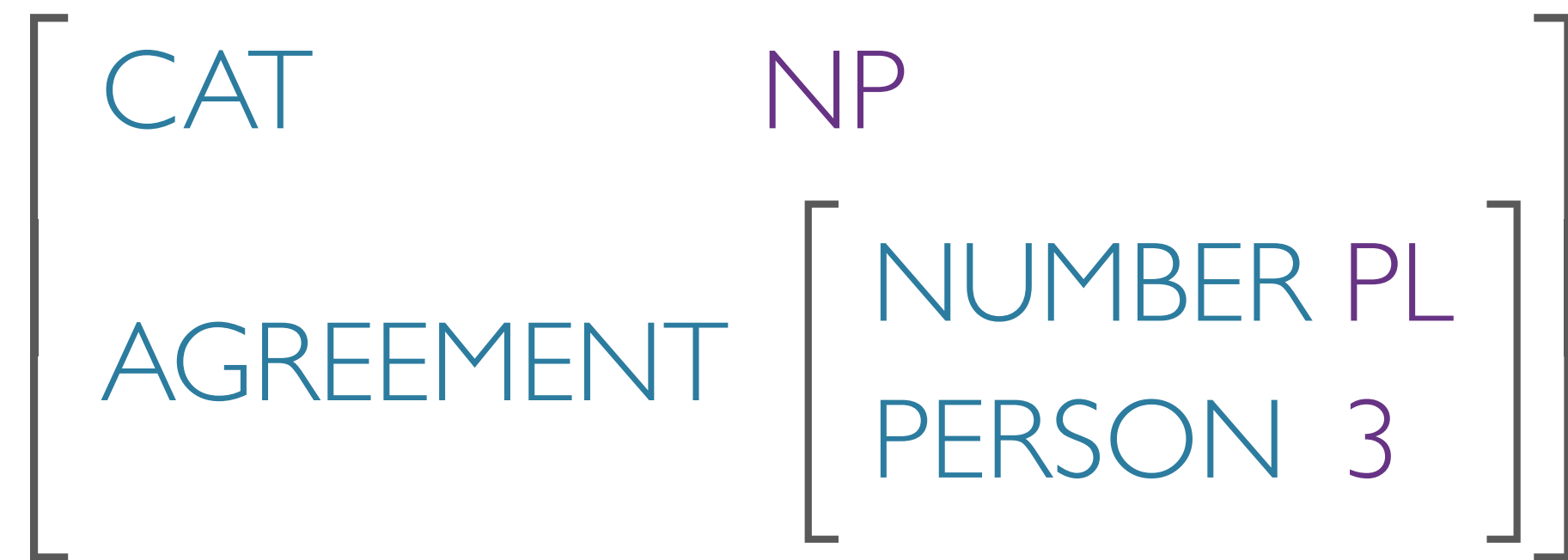
(A) $\left[\begin{array}{ll} \text{NUMBER PL} \\ \text{PERSON 3} \end{array} \right]$

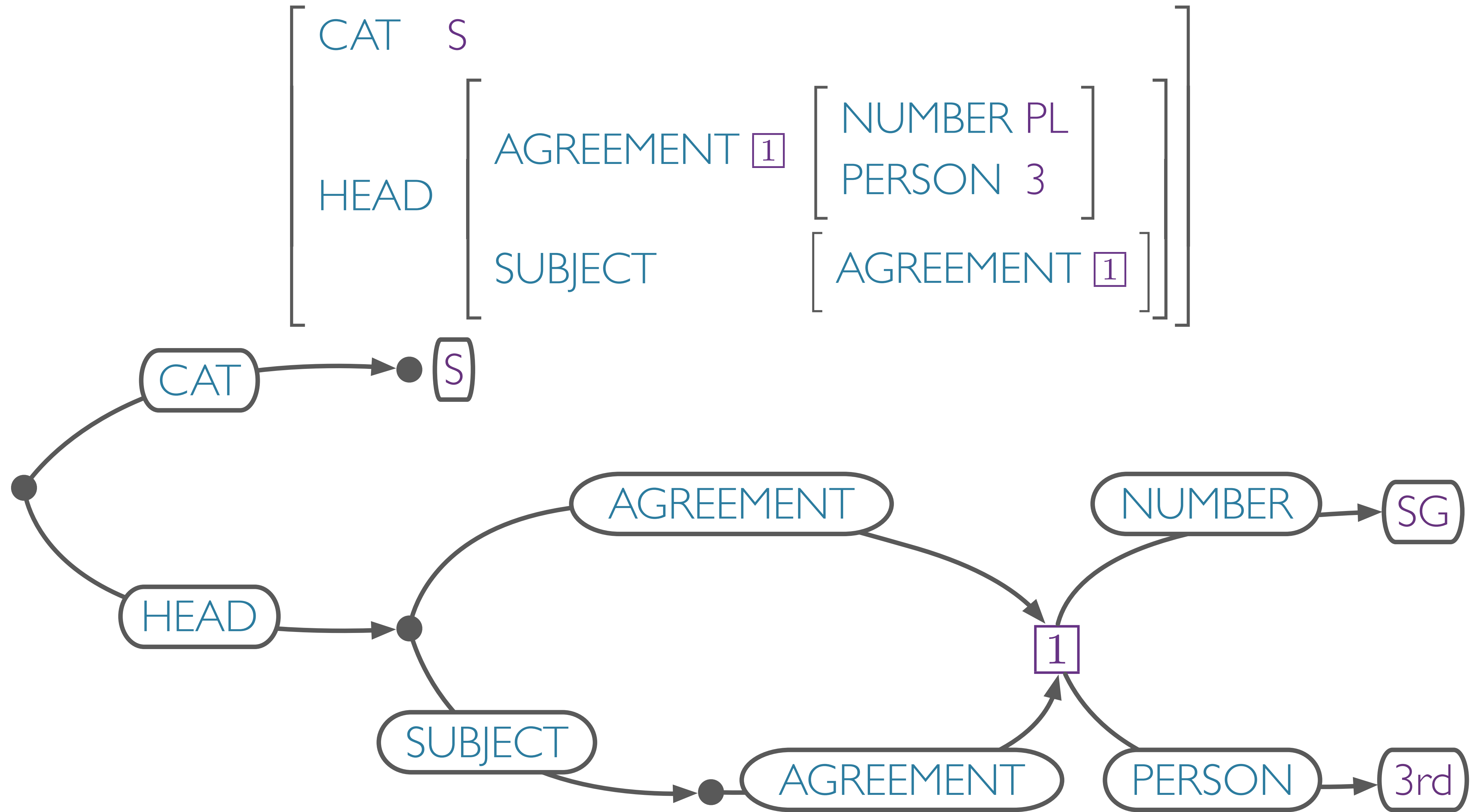
(C) $\left[\begin{array}{ll} \text{CAT} & \text{NP} \\ \text{AGREEMENT} & \left[\begin{array}{ll} \text{NUMBER PL} \\ \text{PERSON 3} \end{array} \right] \end{array} \right]$

(B) $\left[\begin{array}{ll} \text{CAT} & \text{NP} \\ \text{NUMBER PL} \\ \text{PERSON 3} \end{array} \right]$

(D) $\left[\begin{array}{ll} \text{CAT} & \text{S} \\ \text{HEAD} & \left[\begin{array}{ll} \text{AGREEMENT } \boxed{1} & \left[\begin{array}{ll} \text{NUMBER PL} \\ \text{PERSON 3} \end{array} \right] \\ \text{SUBJECT} & \left[\begin{array}{l} \text{AGREEMENT } \boxed{1} \end{array} \right] \end{array} \right] \end{array} \right]$

AVM vs. DAG





Using Feature Structures

- Feature Structures provide formalism to specify constraints
- ...but how to apply the constraints?
- *Unification*

Unification:



- Two key roles:
 - Merge compatible feature structures
 - Reject incompatible feature structures

Unification:



- Two key roles:
 - Merge compatible feature structures
 - Reject incompatible feature structures
- Two structures can unify if:
 - Feature structures *match where both have values*
 - Feature structures *differ only where one value is missing or underspecified*
 - Missing or underspecified values are filled with constraints of other

Unification:



- Two key roles:
 - Merge compatible feature structures
 - Reject incompatible feature structures
- Two structures can unify if:
 - Feature structures *match where both have values*
 - Feature structures *differ only where one value is missing or underspecified*
 - Missing or underspecified values are filled with constraints of other
- Result of unification incorporates constraints of both

Subsumption

- Less specific feature structure ***subsumes*** more specific feature structure

Subsumption

- Less specific feature structure **subsumes** more specific feature structure
- FS F subsumes FS G iff:
 - For every feature x in F , $F(x)$ subsumes $G(x)$
 - for all paths p and q in F s.t. $F(p)=F(q)$, $G(p)=G(q)$

Subsumption

- Less specific feature structure **subsumes** more specific feature structure
- FS F subsumes FS G iff:
 - For every feature x in F , $F(x)$ subsumes $G(x)$
 - for all paths p and q in F s.t. $F(p)=F(q)$, $G(p)=G(q)$

- Examples:

- $A = \begin{bmatrix} \text{NUMBER SG} \end{bmatrix}$ $B = \begin{bmatrix} \text{PERSON 3} \end{bmatrix}$
 $C = \begin{bmatrix} \text{NUMBER SG} \\ \text{PERSON 3} \end{bmatrix}$

Subsumption

- Less specific feature structure **subsumes** more specific feature structure
- FS F subsumes FS G iff:
 - For every feature x in F , $F(x)$ subsumes $G(x)$
 - for all paths p and q in F s.t. $F(p)=F(q)$, $G(p)=G(q)$

- Examples:

- $A = \begin{bmatrix} \text{NUMBER SG} \end{bmatrix}$

$$B = \begin{bmatrix} \text{PERSON 3} \end{bmatrix}$$

- A **subsumes** C

$$C = \begin{bmatrix} \text{NUMBER SG} \\ \text{PERSON 3} \end{bmatrix}$$

Subsumption

- Less specific feature structure **subsumes** more specific feature structure
- FS F subsumes FS G iff:
 - For every feature x in F , $F(x)$ subsumes $G(x)$
 - for all paths p and q in F s.t. $F(p)=F(q)$, $G(p)=G(q)$

- Examples:

- $A = \begin{bmatrix} \text{NUMBER SG} \end{bmatrix}$
 $C = \begin{bmatrix} \text{NUMBER SG} \\ \text{PERSON 3} \end{bmatrix}$

$$B = \begin{bmatrix} \text{PERSON 3} \end{bmatrix}$$

- A **subsumes** C
- B **subsumes** C

Subsumption

- Less specific feature structure **subsumes** more specific feature structure
- FS F subsumes FS G iff:
 - For every feature x in F , $F(x)$ subsumes $G(x)$
 - for all paths p and q in F s.t. $F(p)=F(q)$, $G(p)=G(q)$
- Examples:
 - $A = \begin{bmatrix} \text{NUMBER SG} \end{bmatrix}$ $B = \begin{bmatrix} \text{PERSON 3} \end{bmatrix}$
 - A **subsumes** C
 - B **subsumes** C
 - B & A **don't subsume**
 - $C = \begin{bmatrix} \text{NUMBER SG} \\ \text{PERSON 3} \end{bmatrix}$

Unification Examples

- Identical

$$\left[\text{NUMBER SG} \right] \sqcup \left[\text{NUMBER SG} \right] = \left[\text{NUMBER SG} \right]$$

Unification Examples

- Identical

$$\left[\text{NUMBER SG} \right] \sqcup \left[\text{NUMBER SG} \right] = \left[\text{NUMBER SG} \right]$$

- Underspecified

$$\left[\text{NUMBER SG} \right] \sqcup \left[\quad \right] = \left[\text{NUMBER SG} \right]$$

Unification Examples

- Identical

$$\begin{bmatrix} \text{NUMBER SG} \end{bmatrix} \sqcup \begin{bmatrix} \text{NUMBER SG} \end{bmatrix} = \begin{bmatrix} \text{NUMBER SG} \end{bmatrix}$$

- Underspecified

$$\begin{bmatrix} \text{NUMBER SG} \end{bmatrix} \sqcup \begin{bmatrix} \quad \end{bmatrix} = \begin{bmatrix} \text{NUMBER SG} \end{bmatrix}$$

- Different Specs

$$\begin{bmatrix} \text{NUMBER SG} \end{bmatrix} \sqcup \begin{bmatrix} \text{PERSON 3} \end{bmatrix} = \begin{bmatrix} \text{NUMBER SG} \\ \text{PERSON 3} \end{bmatrix}$$

Unification Examples

- Identical

$$\begin{bmatrix} \text{NUMBER SG} \end{bmatrix} \sqcup \begin{bmatrix} \text{NUMBER SG} \end{bmatrix} = \begin{bmatrix} \text{NUMBER SG} \end{bmatrix}$$

- Underspecified

$$\begin{bmatrix} \text{NUMBER SG} \end{bmatrix} \sqcup \begin{bmatrix} \quad \end{bmatrix} = \begin{bmatrix} \text{NUMBER SG} \end{bmatrix}$$

- Different Specs

$$\begin{bmatrix} \text{NUMBER SG} \end{bmatrix} \sqcup \begin{bmatrix} \text{PERSON 3} \end{bmatrix} = \begin{bmatrix} \text{NUMBER SG} \\ \text{PERSON 3} \end{bmatrix}$$

- Conflicting Specs

$$\begin{bmatrix} \text{NUMBER SG} \end{bmatrix} \sqcup \begin{bmatrix} \text{NUMBER PL} \end{bmatrix} = \emptyset$$

Larger Unification Example

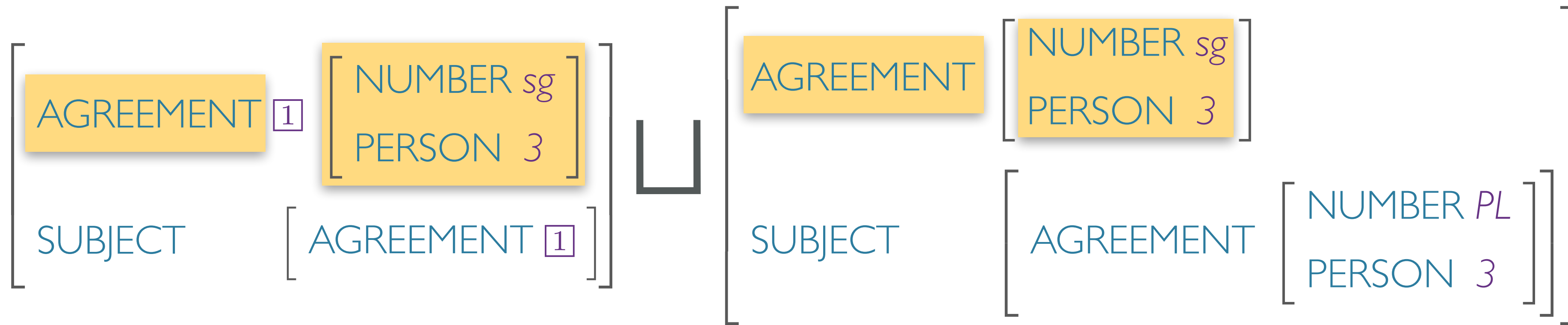
$$\left[\begin{array}{l} \text{AGREEMENT } [1] \\ \text{SUBJECT } \left[\text{AGREEMENT } [1] \right] \end{array} \right] \sqcup \left[\text{SUBJECT } \left[\text{AGREEMENT } \left[\begin{array}{l} \text{PERSON } 3 \\ \text{NUMBER } \text{SG} \end{array} \right] \right] \right] =$$

$$\left[\begin{array}{l} \text{AGREEMENT } [1] \\ \text{SUBJECT } \left[\text{AGREEMENT } [1] \left[\begin{array}{l} \text{PERSON } 3 \\ \text{NUMBER } \text{SG} \end{array} \right] \right] \end{array} \right]$$

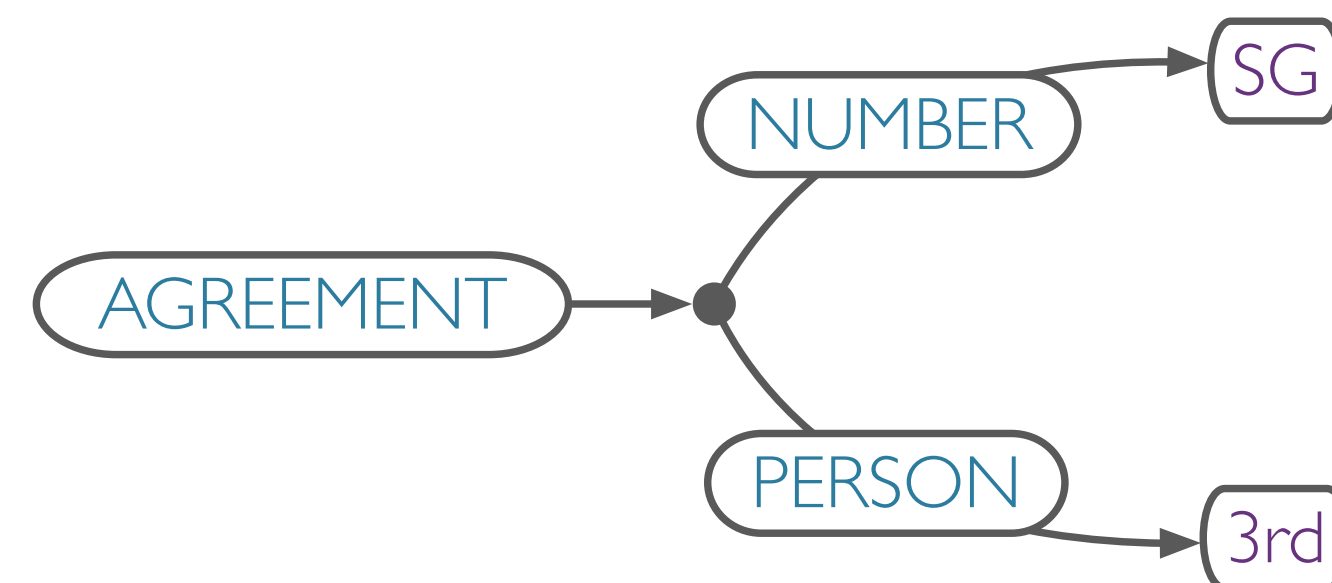
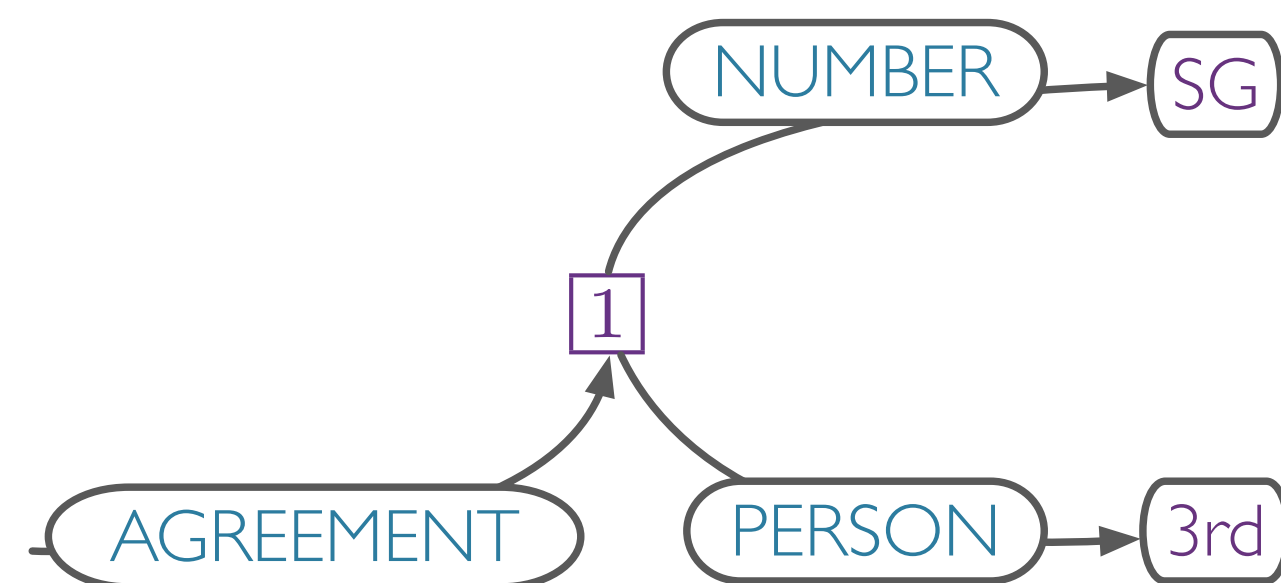
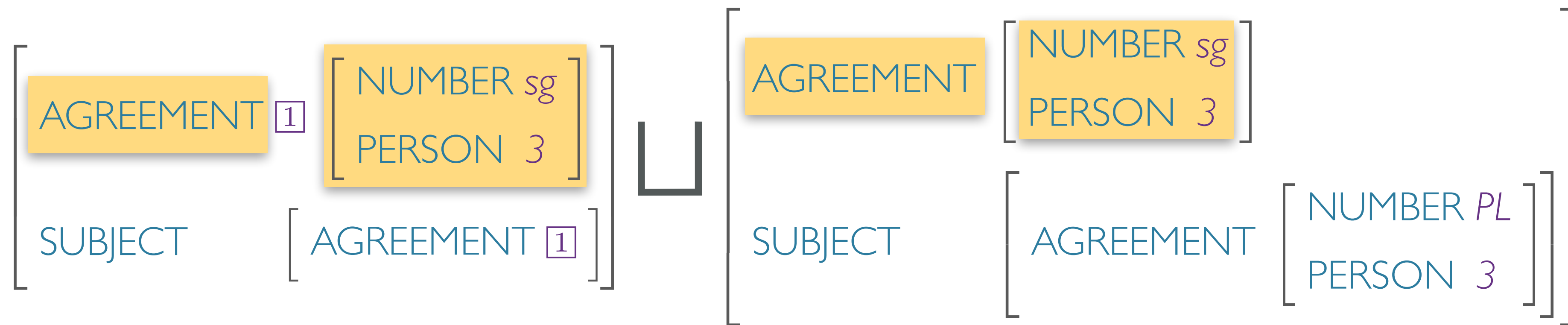
One More Unification Example

$$\left[\begin{array}{l} \text{AGREEMENT } \boxed{1} \left[\begin{array}{l} \text{NUMBER } sg \\ \text{PERSON } 3 \end{array} \right] \\ \text{SUBJECT} \left[\text{AGREEMENT } \boxed{1} \right] \end{array} \right] \sqcup \left[\begin{array}{l} \text{AGREEMENT} \left[\begin{array}{l} \text{NUMBER } sg \\ \text{PERSON } 3 \end{array} \right] \\ \text{SUBJECT} \left[\text{AGREEMENT} \left[\begin{array}{l} \text{NUMBER } PL \\ \text{PERSON } 3 \end{array} \right] \right] \end{array} \right]$$

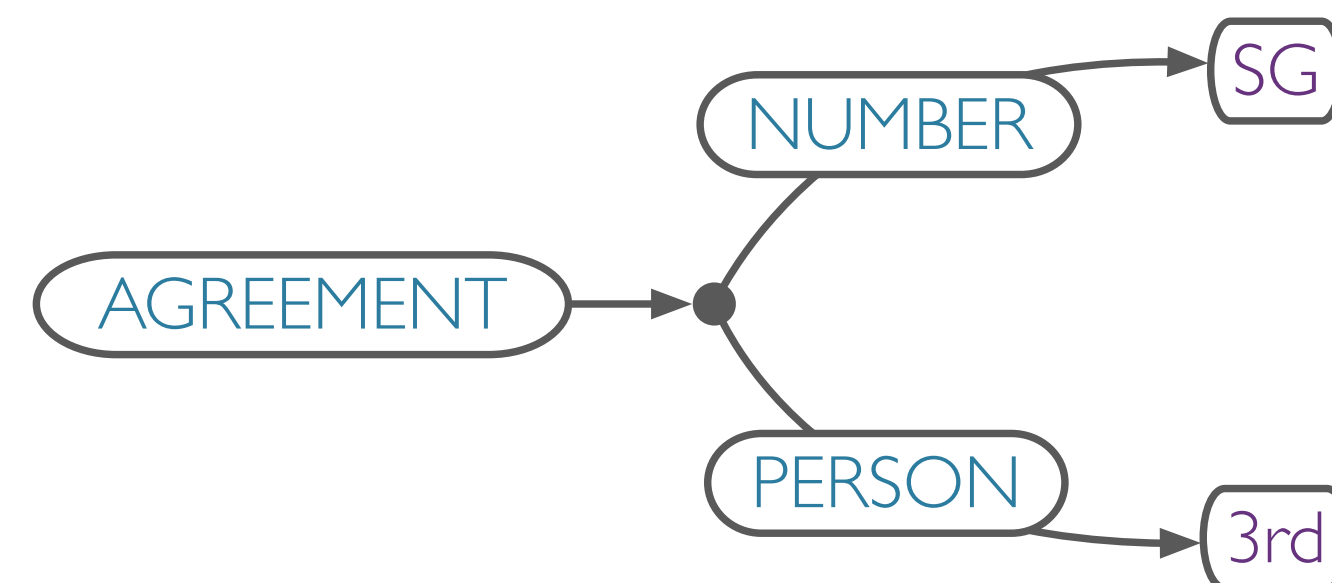
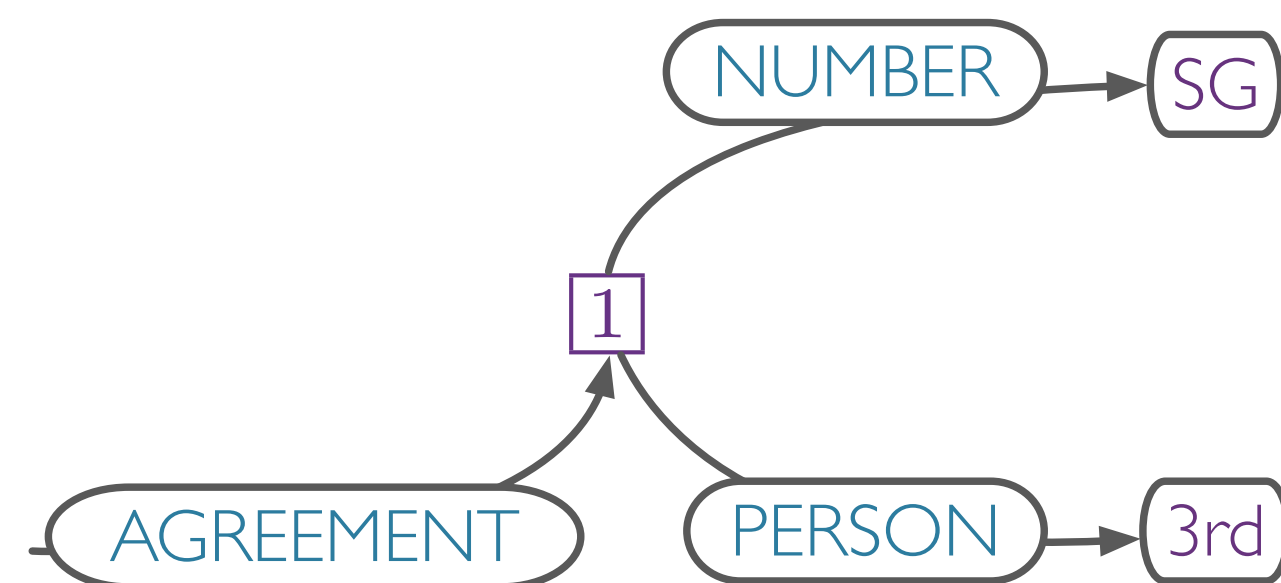
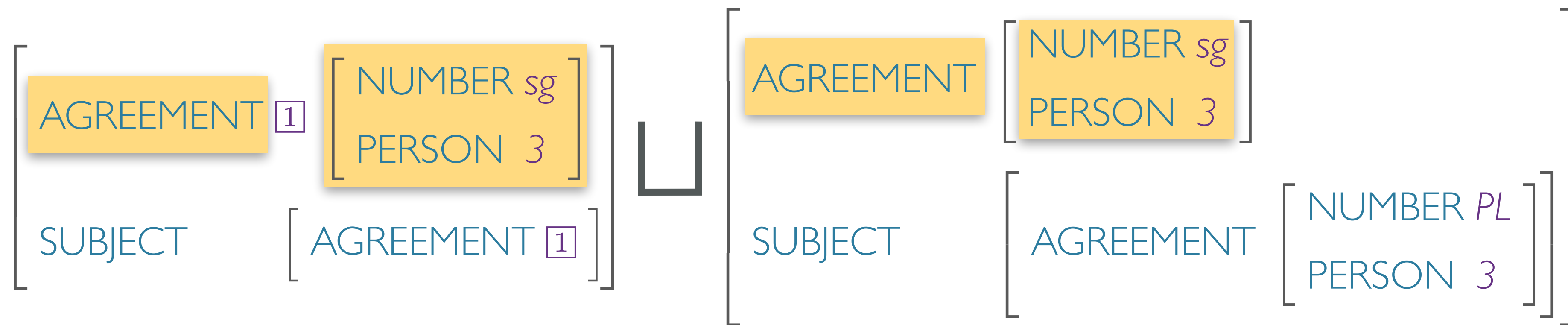
One More Unification Example



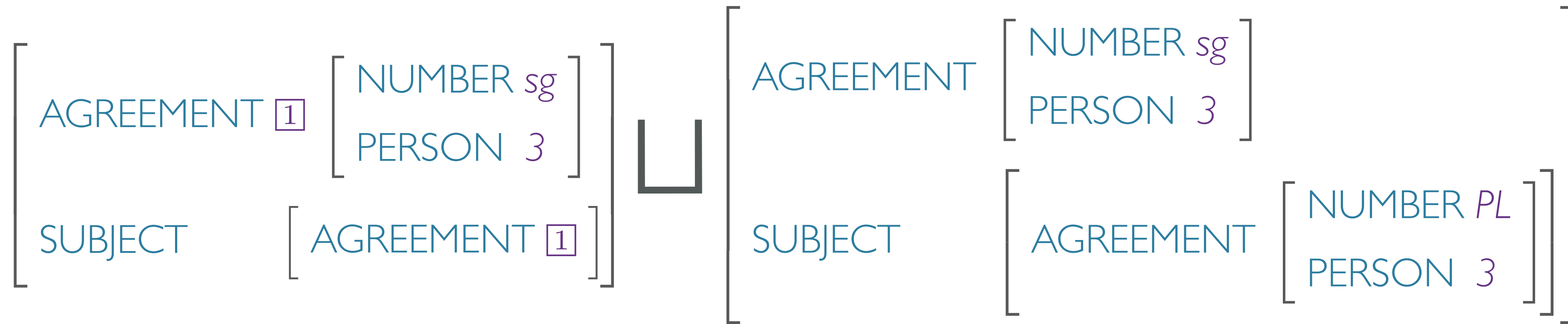
One More Unification Example



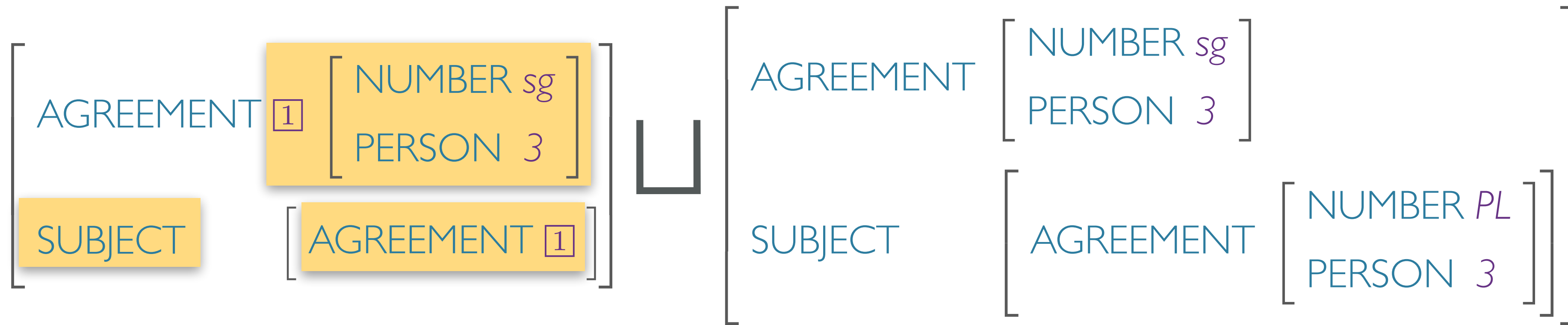
One More Unification Example



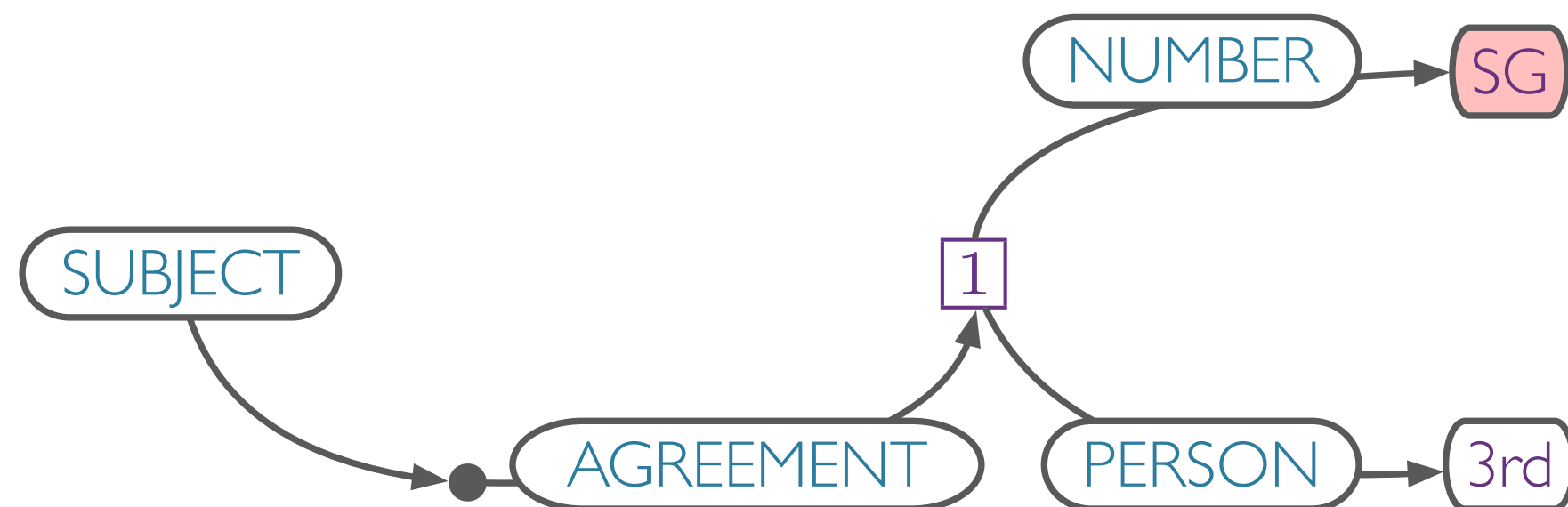
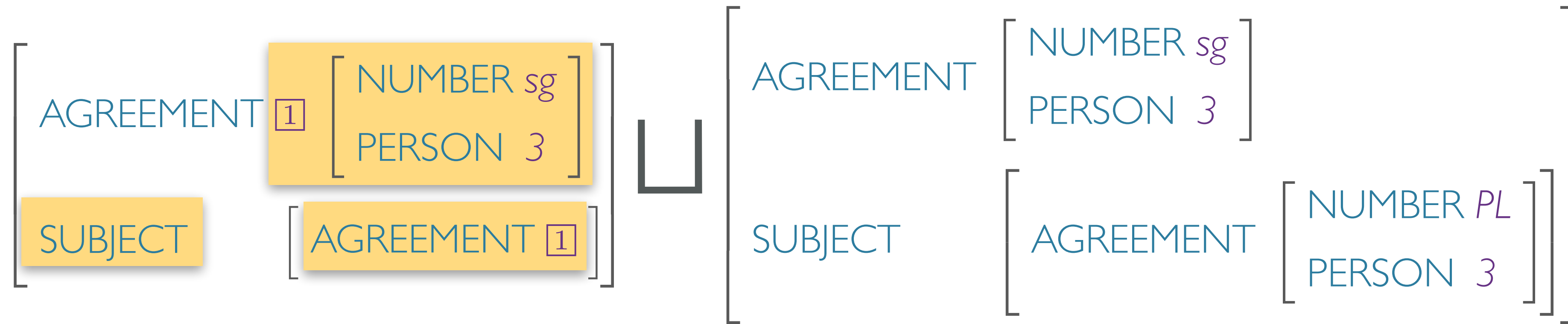
Unification



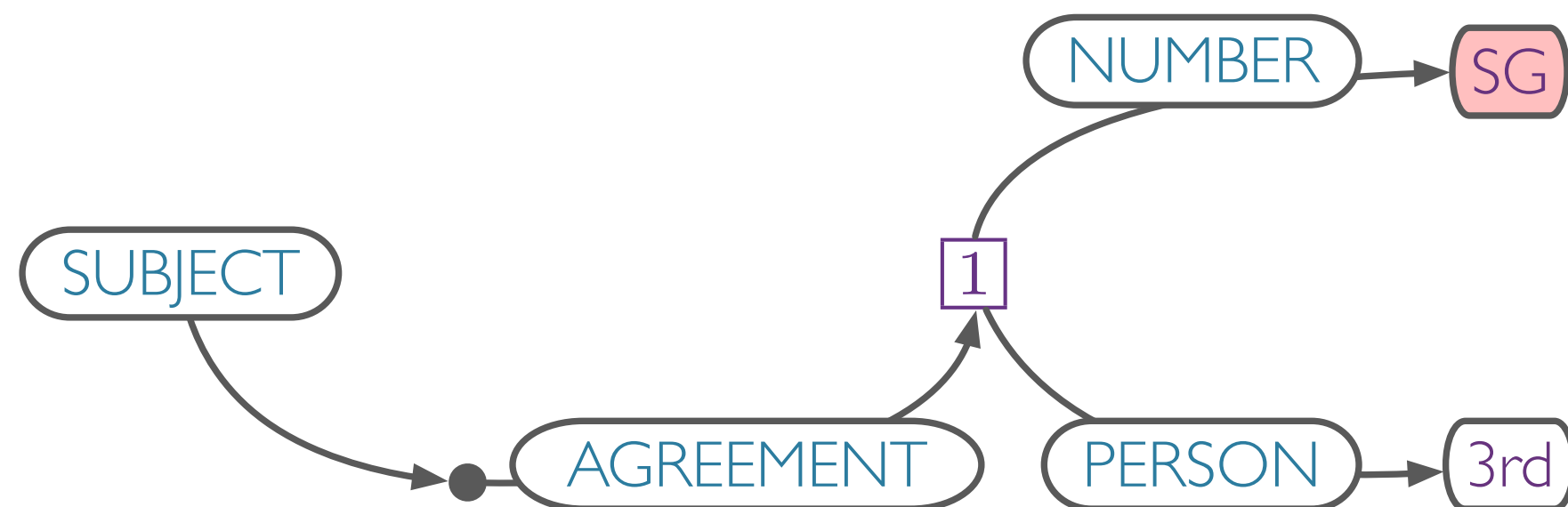
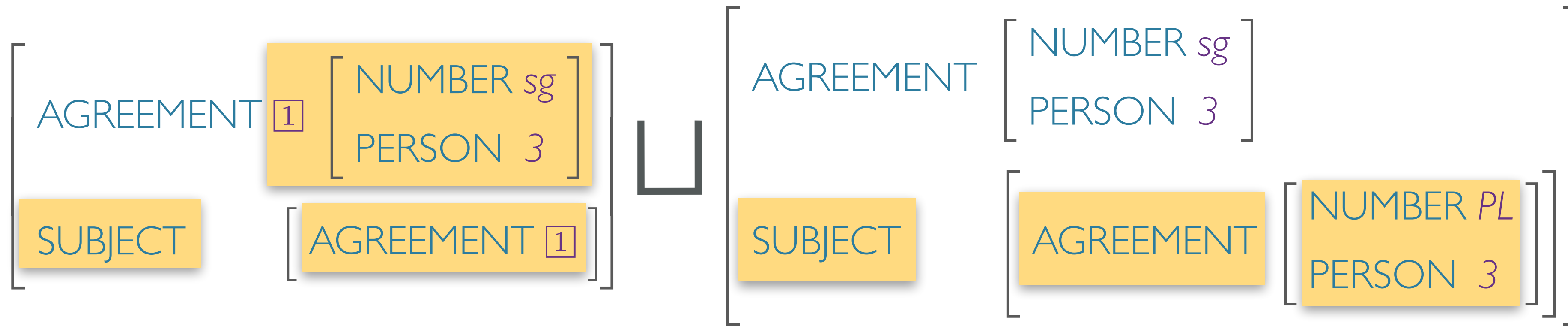
Unification



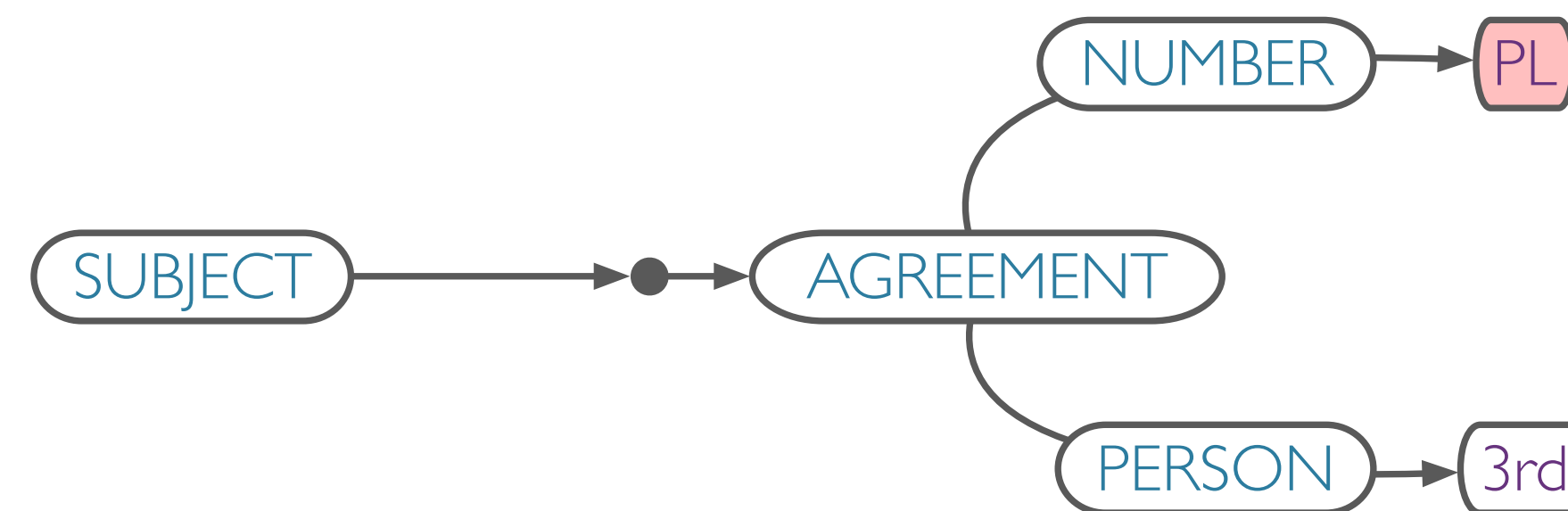
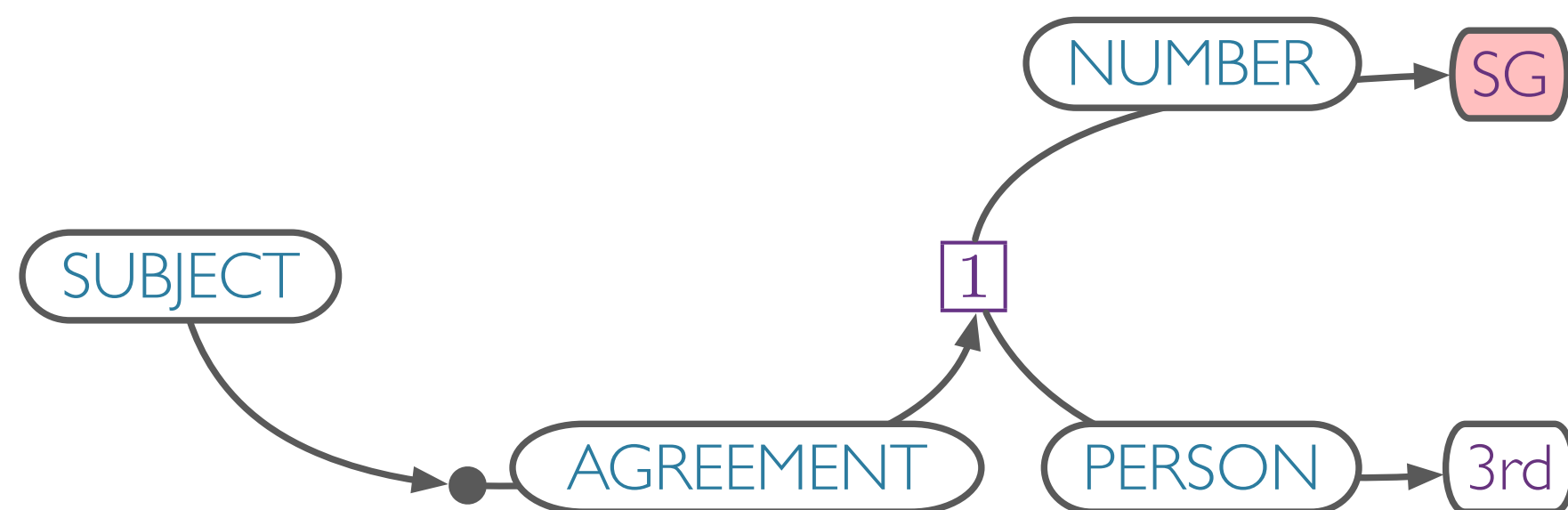
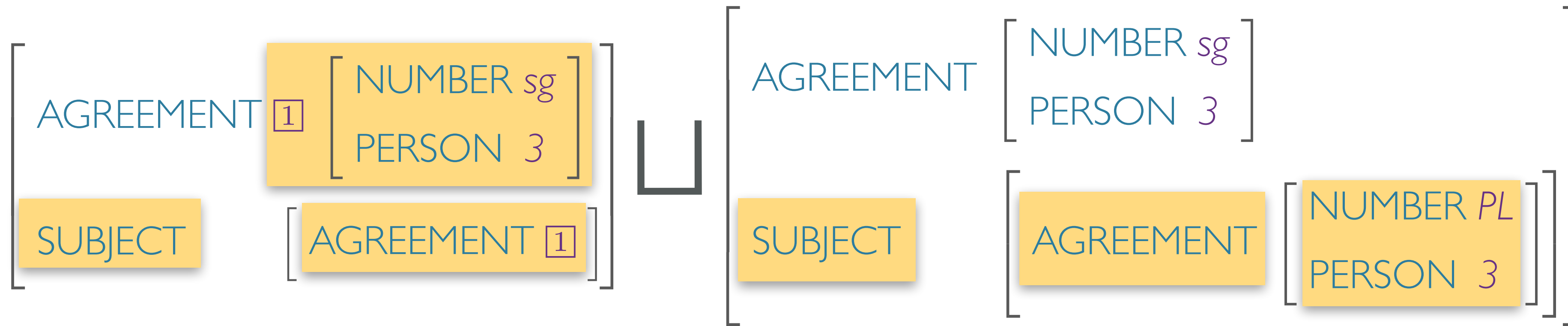
Unification



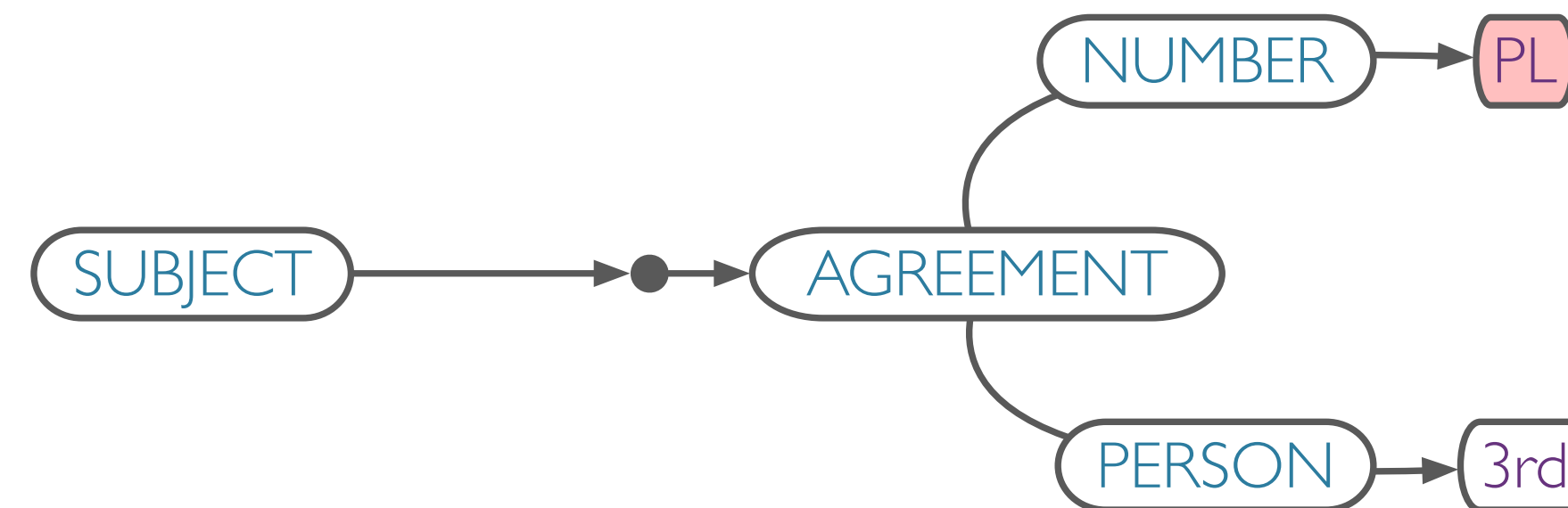
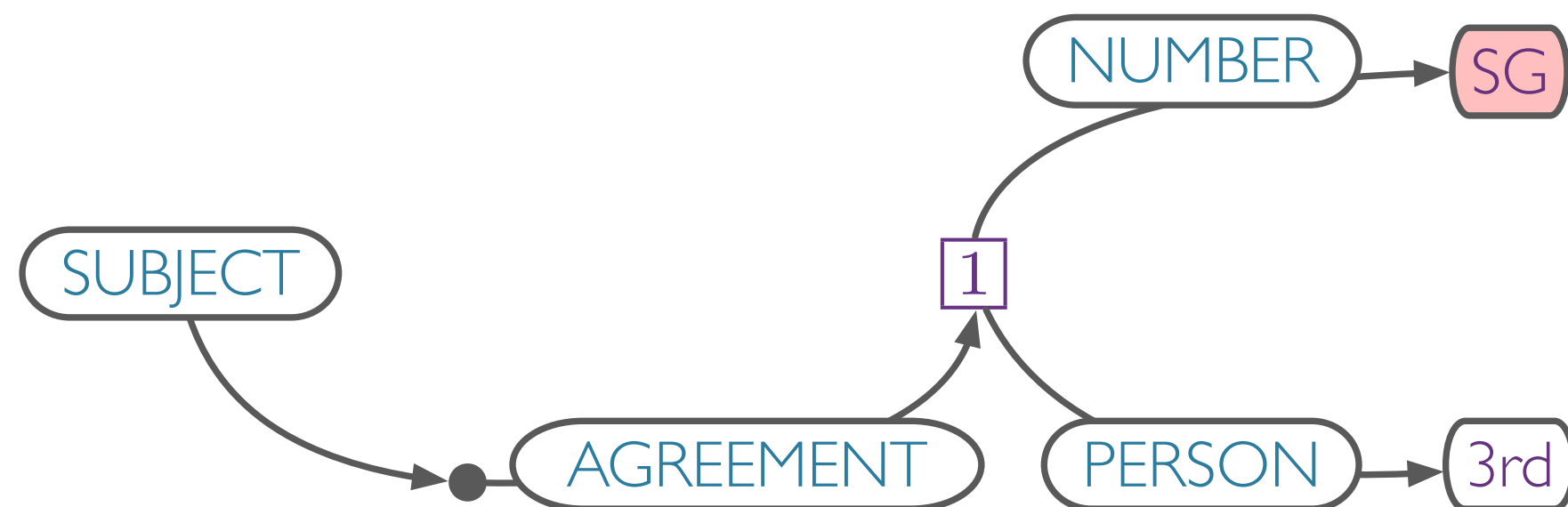
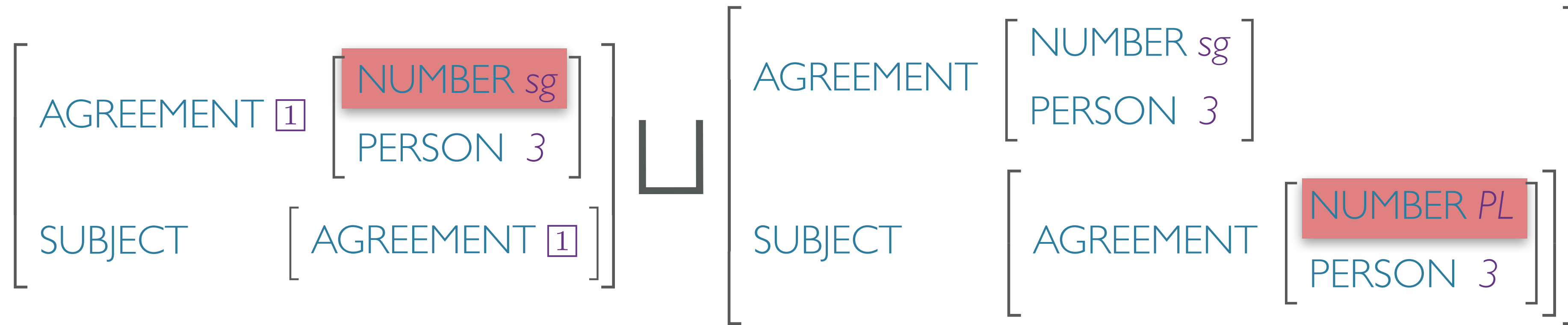
Unification



Unification

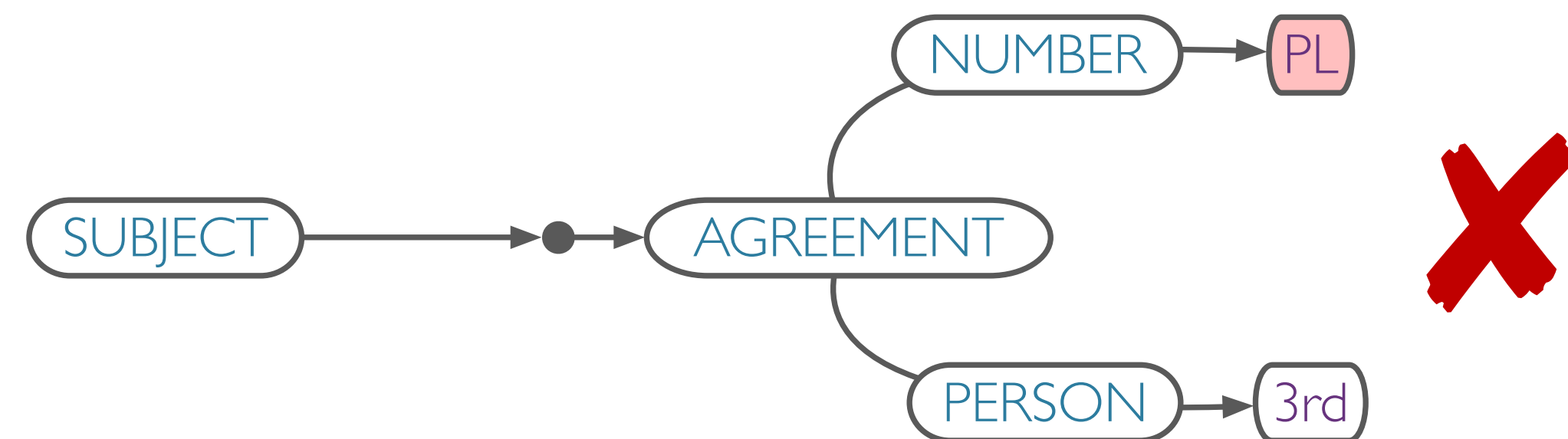
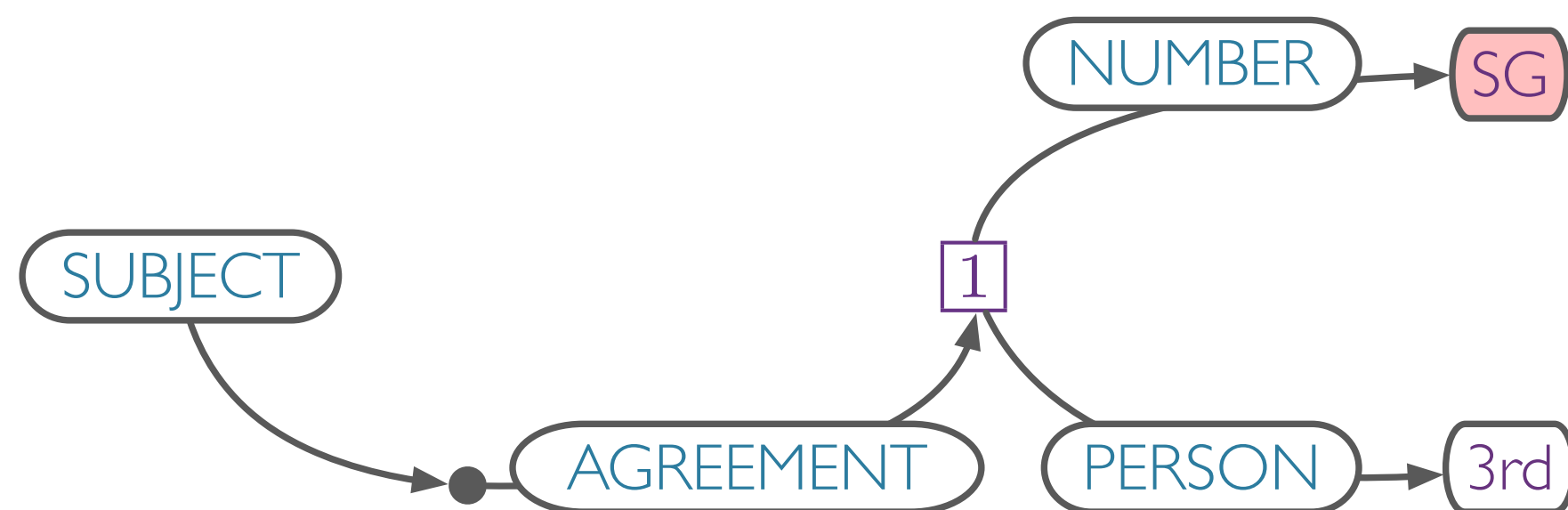


Unification



Unification

$$\left[\begin{array}{l} \text{AGREEMENT } [1] \\ \text{SUBJECT} \end{array} \left[\begin{array}{l} \text{NUMBER } sg \\ \text{PERSON } 3 \\ \text{AGREEMENT } [1] \end{array} \right] \right] \sqcup \left[\begin{array}{l} \text{AGREEMENT} \\ \text{SUBJECT} \end{array} \left[\begin{array}{l} \text{NUMBER } sg \\ \text{PERSON } 3 \\ \text{AGREEMENT} \left[\begin{array}{l} \text{NUMBER } PL \\ \text{PERSON } 3 \end{array} \right] \end{array} \right] \right] = \emptyset \text{ Failure!}$$



Rule Representation

- $\beta \rightarrow \beta_1 \dots \beta_n$
 $\{set\ of\ constraints\}$ $\langle \beta_i feature\ path \rangle = Atomic\ value \mid \langle \beta_j feature\ path \rangle$
- $PRON \rightarrow 'he'$

Rule Representation

- $\beta \rightarrow \beta_1 \dots \beta_n$
 $\{set\ of\ constraints\}$ $\langle \beta_i feature\ path \rangle = Atomic\ value \mid \langle \beta_j feature\ path \rangle$
- $PRON \rightarrow 'he'$

$\langle PRON$

Pron

Rule Representation

- $\beta \rightarrow \beta_1 \dots \beta_n$
 $\{\text{set of constraints}\}$ $\langle \beta_i \text{ feature path} \rangle = \text{Atomic value} \mid \langle \beta_j \text{ feature path} \rangle$
- $PRON \rightarrow \text{'he'}$

$\langle PRON \text{ AGREEMENT} \rangle$

Pron

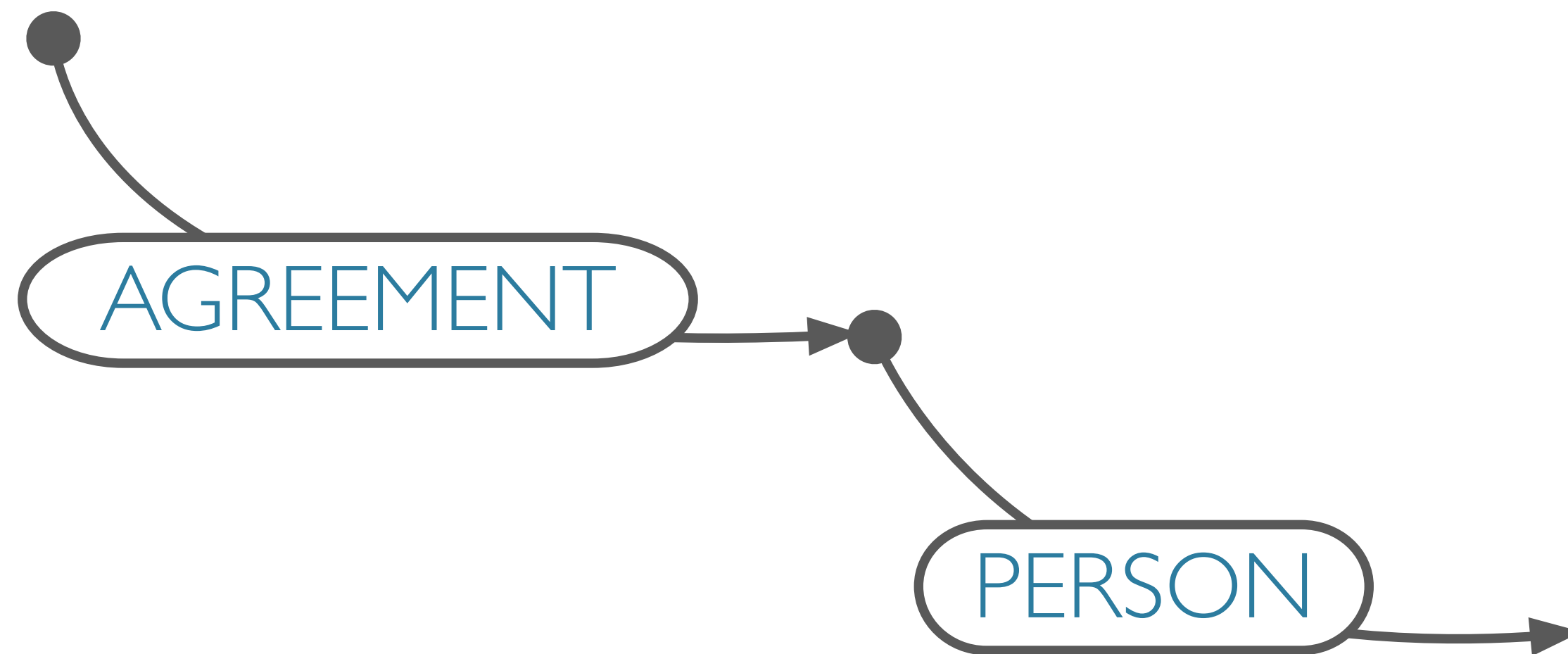


Rule Representation

- $\beta \rightarrow \beta_1 \dots \beta_n$
 $\{\text{set of constraints}\}$ $\langle \beta_i \text{ feature path} \rangle = \text{Atomic value} \mid \langle \beta_j \text{ feature path} \rangle$
- $PRON \rightarrow \text{'he'}$

$\langle PRON \text{ AGREEMENT PERSON} \rangle$

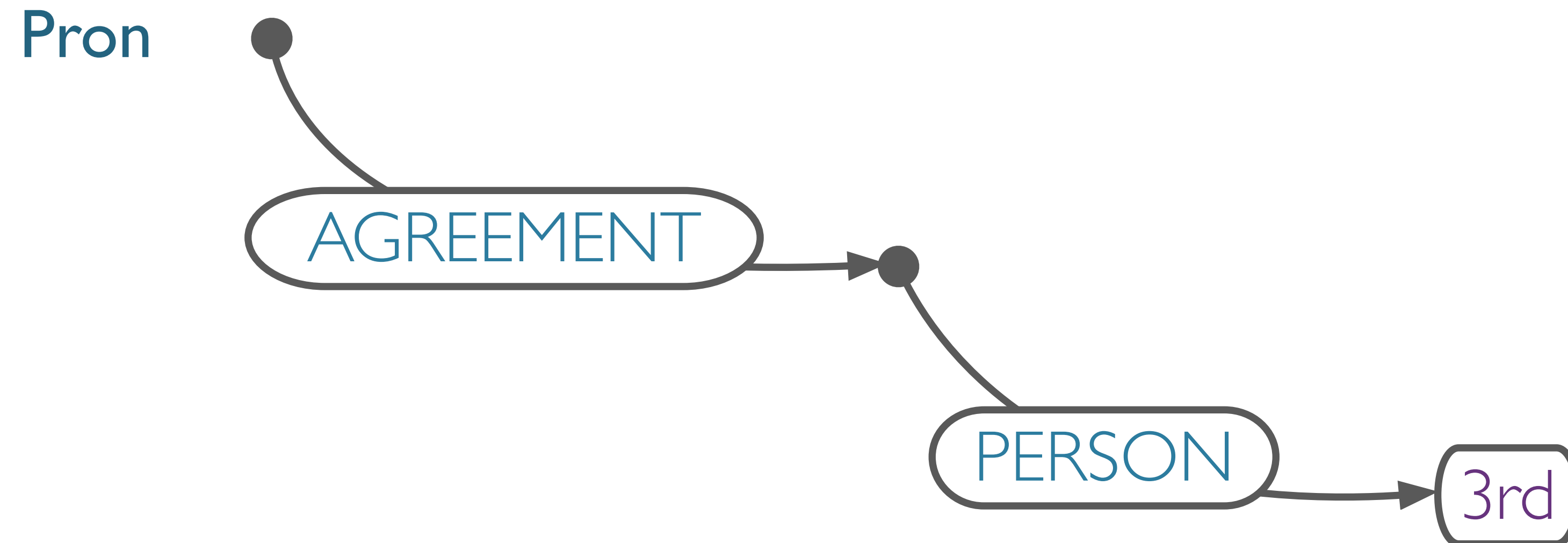
Pron



Rule Representation

- $\beta \rightarrow \beta_1 \dots \beta_n$
 $\{\text{set of constraints}\}$ $\langle \beta_i \text{ feature path} \rangle = \text{Atomic value} \mid \langle \beta_j \text{ feature path} \rangle$
- $PRON \rightarrow \text{'he'}$

$\langle PRON \text{ AGREEMENT PERSON} \rangle = 3rd$

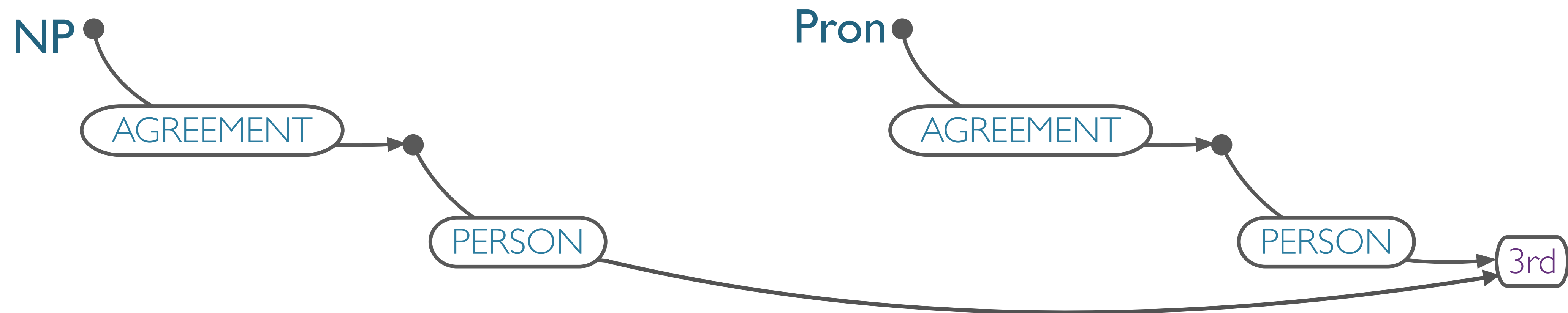


Rule Representation

- $\beta \rightarrow \beta_1 \dots \beta_n$
 $\{\text{set of constraints}\}$ $\langle \beta_i \text{ feature path} \rangle = \text{Atomic value} \mid \langle \beta_j \text{ feature path} \rangle$

- $NP \rightarrow PRON$

$\langle \textbf{NP AGREEMENT PERSON} \rangle = \langle \textbf{PRON AGREEMENT PERSON} \rangle$

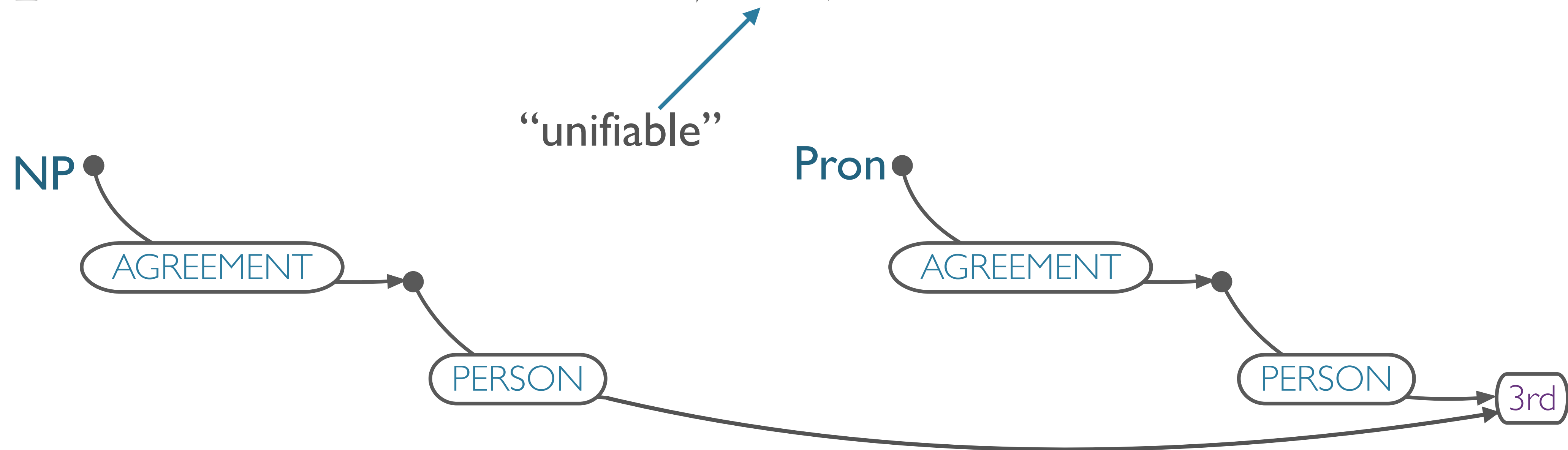


Rule Representation

- $\beta \rightarrow \beta_1 \dots \beta_n$
 $\{\text{set of constraints}\}$ $\langle \beta_i \text{ feature path} \rangle = \text{Atomic value} \mid \langle \beta_j \text{ feature path} \rangle$

- $NP \rightarrow PRON$

$\langle \textbf{NP AGREEMENT PERSON} \rangle = \langle \textbf{PRON AGREEMENT PERSON} \rangle$



Agreement with Heads and Features

- $\beta \rightarrow \beta_1 \dots \beta_n$
 $\{set\ of\ constraints\}$ $\langle \beta_i\ feature\ path \rangle = Atomic\ value \mid \langle \beta_j\ feature\ path \rangle$

$S \rightarrow NP\ VP$

$\langle NP\ AGREEMENT \rangle = \langle VP\ AGREEMENT \rangle$

$Det \rightarrow this$

$\langle Det\ AGREEMENT\ NUMBER \rangle = sg$

$S \rightarrow Aux\ NP\ VP$

$\langle Aux\ AGREEMENT \rangle = \langle NP\ AGREEMENT \rangle$

$Det \rightarrow these$

$\langle Det\ AGREEMENT\ NUMBER \rangle = pl$

$NP \rightarrow Det\ Nominal$

$\langle Det\ AGREEMENT \rangle = \langle Nominal\ AGREEMENT \rangle$

$\langle NP\ AGREEMENT \rangle = \langle Nominal\ AGREEMENT \rangle$

$Verb \rightarrow serve$

$\langle Verb\ AGREEMENT\ NUMBER \rangle = pl$

$Aux \rightarrow does$

$\langle AUX\ AGREEMENT\ NUMBER \rangle = sg$

$\langle AUX\ AGREEMENT\ PERSON \rangle = 3rd$

$Noun \rightarrow flight$

$\langle Noun\ AGREEMENT\ NUMBER \rangle = sg$

Simple Feature Grammars in NLTK

- $S \rightarrow NP VP$

Simple Feature Grammars

- $S \rightarrow NP[NUM=?n] VP[NUM=?n]$
- $NP[NUM=?n] \rightarrow N[NUM=?n]$
- $NP[NUM=?n] \rightarrow PropN[NUM=?n]$
- $NP[NUM=?n] \rightarrow Det[NUM=?n] N[NUM=?n]$
- $Det[NUM=sg] \rightarrow 'this' \mid 'every'$
- $Det[NUM=pl] \rightarrow 'these' \mid 'all'$
- $N[NUM=sg] \rightarrow 'dog' \mid 'girl' \mid 'car' \mid 'child'$
- $N[NUM=pl] \rightarrow 'dogs' \mid 'girls' \mid 'cars' \mid 'children'$

Parsing with Features

```
>>> cp = load_parser('grammars/book_grammars/  
feat0.fcfg')  
>>> for tree in cp.parse(tokens):  
...     print(tree)
```

```
(S[ ] (NP[NUM='sg']  
  (PropN[NUM='sg'] Kim))  
  (VP[NUM='sg', TENSE='pres']  
    (TV[NUM='sg', TENSE='pres'] likes)  
    (NP[NUM='pl'] (N[NUM='pl'] children))))
```

Feature Applications

- Subcategorization
 - Verb-Argument constraints
 - Number, type, characteristics of args
 - e.g. is the subject *animate*?
 - Also adjectives, nouns
- Long-distance dependencies
 - e.g. filler–gap relations in wh-questions
 - “Which flight do you want me to have the travel agent book?”

Morphosyntactic Features

- Grammatical feature that influences morphological or syntactic behavior
 - English:
 - Number:
 - Dog, dogs
 - Person:
 - am; are; is
 - Case:
 - I / me; he / him; etc.

Semantic Features

- Grammatical features that influence semantic (meaning) behavior of associated units
- E.g.:
 - *?The rocks slept. ? Colorless green ideas sleep furiously.*
- Many proposed:
 - Animacy: +/–
 - Human: +/–
 - Adult: +/–
 - Liquid: +/–

Aspect (J&M 17.4.2)

- *The climber [hiked] [for six hours].*

Aspect (J&M 17.4.2)

- *The climber [hiked] [for six hours].*
- *The climber [hiked] [on Saturday].*

Aspect (J&M 17.4.2)

- *The climber [hiked] [for six hours].*
- *The climber [hiked] [on Saturday].*
- *The climber [reached the summit] [on Saturday].*

Aspect (J&M 17.4.2)

- *The climber [hiked] [for six hours].*
- *The climber [hiked] [on Saturday].*
- *The climber [reached the summit] [on Saturday].*
- **The climber [reached the summit] [for six hours].*

Aspect (J&M 17.4.2)

- *The climber [hiked] [for six hours].*
 - *The climber [hiked] [on Saturday].*
 - *The climber [reached the summit] [on Saturday].*
 - **The climber [reached the summit] [for six hours].*
-
- Contrast:
 - *Achievement* (in an instant) vs *activity* (for a time)